MULTI-AGENT BASED CONTROL OF LARGE-SCALE COMPLEX SYSTEMS EMPLOYING DISTRIBUTED DYNAMIC INFERENCE ENGINE

A Dissertation
Presented to
The Academic Faculty

by

Daili Zhang

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy

School of Aerospace Engineering
Georgia Institute of Technology
May 2010

MULTI-AGENT BASED CONTROL OF

LARGE-SCALE COMPLEX SYSTEMS EMPLOYING

DISTRIBUTED DYNAMIC INFERENCE ENGINE

Approved by:

Dr. Dimitri N. Mavris, Advisor
School of Aereospace Engineering
Georgia Institute of Technology

Dr. Mark Costello
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Vitali Volovoi
School of Aerospace Engineering
Georgia Institute of Technology

Mr. Frank T. Ferrese
Lead Engineer
Office of Navy Research

Dr. Neil R. Weston
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 26th March, 2010

ACKNOWLEDGMENTS

The work presented in this dissertation would not have been possible were it not for the invaluable help and support offered to me by many people. I would like to express my sincere gratitude to all of them.

First and foremost, I would like to thank my advisor and committee chair Dr. Dimitri Mavris for his continual encouragement, support, guidance, and inspiration over the years. His broad knowledge, unique insight into system designs, enthusiasm for new ideas, outstanding leadership skills, and great personality make him a very capable advisor and I cannot express how much I've benefited both academically and generally in life.

I would also like to express my great gratitude to Dr. Neil Weston, Dr. Vitali Volovoi and Dr. Mark Costello for their kindness, patience, invaluable suggestions, and guidance. And I am particularly grateful to the Office of Naval Research (ONR) for funding and supporting this work, and to Mr. Frank Ferrese and Mr. Anthony Seman for their insightful opinions.

I have immensely enjoyed my time in the Aerospace System Design Lab (ASDL). ASDL has provided me a friendly, enjoyable and supportive environment. Special thanks to my labmates: Santiago Balestrini, Fairuz Romli, Kyungjin Moon, Hongjun Ran, Young-Ki Lee, Cedric Justin, Bassem Nairouz, Matt Hoepfer, Kybeom Kwon, Michael Balchanos, Dave Fullmer, Joo Sung Kang, Dane Freeman, etc.

Finally, I would like to thank my parents, my sister, and my friends Jim Brooks, Al LaCour, Yun Li, Bing Xiao, Haohua Xu, Chang Chien Hui-han and Lan Wu for their enduring support and love. Last but not least, I deeply appreciate my dear husband Dr. Nan Jiang for his warm and endless love. He believes in me even when I doubt myself.

Thank you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

3APL	An Abstract Agent Programming Language
ABCtrl	Agent-Based Control
BDI	Belief-Desire-Intention
BK	Boyen-Koller
CA	Contribution Analysis
CA/NA	Completely Accurate, Nearly Autonomous
CBR	Case-Based Reasoning
CHA	Coordinated Hybrid Agent
CWS	Chilled Water System
DAG	Directed Acyclic Graph
DBNs	Dynamic Bayesian Networks
DBP	Distributed Belief Propagation
DDBNs	Distributed Dynamic Bayesian Networks
DDSSS	Distributed D-Sep Set Satisfaction
DKF	Distributed Kalman Filtering
DPFs	Distributed Particle Filters
DPNs	Distributed Perception Networks
DRIS	Distributed Running Intersection Satisfaction
DSNs	Distributed Sensor Networks
DSTO	Distributed Spanning Tree Optimization
EDKFs	Extended Distributed Kalman Filters
FA/C	Functionally Accurate and Cooperative
FIPA	Federal Intelligent Physical Agent
HCIA	Hybrid Control Intelligent Agent
HyMABC	Hybrid Multi-Agent Based Control
IDE	Integrated Development Environment
IRIS	Integrated, Reconfigurable and Intelligent System
JADE	Java Agent DEvelopment
JFLT	Junction Forest Linkage Tree
JNI	Java Native Interface
MA	Multiple Agents

MABC	Multi-Agent Based Control
MSBNs	Multiple Sectioned Bayesian Networks
MSDBNs	Multiple Sectioned Dynamic Bayesian Networks
NTS	National Transportation System
OAK	Open Autonomy Kernel
OOP	Object-Oriented Programming
PLDMs	Prior/Likelihood Decomposable Models
PRS	Procedural Reasoning System
RA	Rockwell Automation
RSAD	Reduced Scale Advanced Demonstrator
TES	Thermo-Electrical System
UAV	Unmanned Aerial Vehicles
WLS	Weighted Least Square

SUMMARY

Increasing societal demand for automation has led to considerable efforts to control large-scale complex systems, especially in the area of autonomous intelligent control methods. The control system of a large-scale complex system needs to satisfy four system level requirements: ***robustness***, ***flexibility, reusability,*** and ***scalability.*** Corresponding to the four system level requirements, there arise four major challenges. First, it is difficult to get accurate and complete information. Second, the system may be physically highly distributed. Third, the system evolves very quickly. Fourth, emergent global behaviors of the system can be caused by small disturbances at the component level.

The Multi-Agent Based Control (MABC) method as an implementation of distributed intelligent control has been the focus of research since the 1970s, in an effort to solve the above-mentioned problems in controlling large-scale complex systems. However, to the author's best knowledge, all MABC systems for large-scale complex systems with significant uncertainties are problem-specific and thus difficult to extend to other domains or larger systems. This situation is partly due to the control architecture of multiple agents being determined by agent to agent coupling and interaction mechanisms. Therefore, the research objective of this dissertation is to ***develop a comprehensive, generalized framework for the control system design of general large-scale complex systems with significant uncertainties***, with the focus on distributed control architecture design and distributed inference engine design.

A Hybrid Multi-Agent Based Control (HyMABC) architecture is proposed by combining hierarchical control architecture and module control architecture with logical replication rings. First, it decomposes a complex system hierarchically; second, it combines the components in the same level as a module, and then designs common interfaces for all of the components in the same module; third, replications are made for critical agents and

are organized into logical rings. This architecture maintains clear guidelines for complexity decomposition and also increases the robustness of the whole system.

Multiple Sectioned Dynamic Bayesian Networks (MSDBNs) as a distributed dynamic probabilistic inference engine, can be embedded into the control architecture to handle uncertainties of general large-scale complex systems. MSDBNs decomposes a large knowledge-based system into many agents. Each agent holds its partial perspective of a large problem domain by representing its knowledge as a Dynamic Bayesian Network (DBN). Each agent accesses local evidence from its corresponding local sensors and communicates with other agents through finite message passing. If the distributed agents can be organized into a tree structure, satisfying the running intersection property and d-sep set requirements, globally consistent inferences are achievable in a distributed way. By using different frequencies for local DBN agent belief updating and global system belief updating, it balances the communication cost with the global consistency of inferences. In this dissertation, a fully factorized Boyen-Koller (BK) approximation algorithm is used for local DBN agent belief updating, and the static Junction Forest Linkage Tree (JFLT) algorithm is used for global system belief updating.

MSDBNs assume a static structure and a stable communication network for the whole system. However, for a real system, sub-Bayesian networks as nodes could be lost, and the communication network could be shut down due to partial damage in the system. Therefore, on-line and automatic MSDBNs structure formation is necessary for making robust state estimations and increasing survivability of the whole system. A Distributed Spanning Tree Optimization (DSTO) algorithm, a Distributed D-Sep Set Satisfaction (DDSSS) algorithm, and a Distributed Running Intersection Satisfaction (DRIS) algorithm are proposed in this dissertation. Combining these three distributed algorithms and a Distributed Belief Propagation (DBP) algorithm in MSDBNs makes state estimations robust to partial damage in the whole system.

Combining the distributed control architecture design and the distributed inference engine design leads to a process of control system design for a general large-scale complex system. As applications of the proposed methodology, the control system design of a simplified ship chilled water system and a notional ship chilled water system have been demonstrated step by step. Simulation results not only show that the proposed methodology gives a clear guideline for control system design for general large-scale complex systems with dynamic and uncertain environment, but also indicate that the combination of MSDBNs and HyMABC can provide excellent performance for controlling general large-scale complex systems.

CHAPTER I

MOTIVATION AND INTRODUCTION

1.1 Large-Scale Complex Systems

In recent decades, there has been a strong interest in control of large-scale complex systems, especially in the area of autonomous intelligent control methods. A complex system is a system composed of interconnected parts that, as a whole, exhibits one or more properties which are not obvious from the properties of the individual parts [1]. For example, the automatic fire suppression system based on ship-wide distributed sensor networks for DD(X) destroyer [2], the networked multiple Unmanned Aerial Vehicles (UAVs) in future combat systems [3], the National Transportation System (NTS) consisting of diverse 'things' which evolve over time with multiple objectives [4], a highly distributed shipboard Chilled Water System (CWS) for support of weapons, surveillance and communication system operations under combat situations [5], a power system covering a large area with unpredictable load demands and component failures [6].

Generally, a complex system exhibits a few important characteristics [7]:

- At any instant of time, there are potentially many different ways in which the system environment can evolve; the system's internal dynamic characteristics can evolve in numerous ways as well (formally, the system environment is nondeterministic, nonlinear and the dynamic characteristics of the system itself are also neither deterministic nor linear). For example, service load requirements under combat

situations should be different from cruise situations for a shipboard cooling system: the weapon subsystem needs more chilled water resource during combat situations.

- At any instant of time, there are potentially many different actions or procedures the system can execute, especially when the information regarding the environment and the system itself has significant uncertainties. Listing all the actions or procedures with their corresponding system and environment states is impractical. For example, a simple system with 10 components, where each component has 5 states, has 5^{10}=9765625 different combinations. Even if some efficient algorithms were developed and applied, it would require significant time to search for matched strategies under different situations.
- At any instant of time, a multiple-objective system with heterogeneous entities potentially has many different objectives that it is required to achieve. For example, a network with unmanned aerial vehicles for surveillance of a specific area should cover a certain area as well as minimize global cost. The actions or procedures that (best) achieve numerous objectives are dependent on the environment state as well as the internal states of the system. Balancing different objectives at different times under different situations is required.
- The environment can only be sensed locally (formally, the system is highly distributed). A localized set of sensor information cannot determine the complete system states which are consistent with global information. Global information consistencies by fusing different localized sensors' information are important for the control system to make correct decisions.
- The rate at which state estimations or strategies planning or actions carry out may be slower than the change of the state of system environment or system dynamics.
- Different parts in the system are connected and have interactions in some way. Small disturbances of one or more entities can be propagated all through the system and

may cause unpredictable system level emergent behaviors and result in a catastrophic crash of the whole system.

1.2 Control of Large-Scale Complex Systems

Controlling a large scale complex system such as described above manually is not only expensive, but also cannot reach certain satisfactory level, because there are limits on a human as a decision maker and a problem solver (especially limits in cognitive processing), which is called bounded rationality [8]. Human beings can be part of the control system, but cannot be the only part. More importantly, complex dynamic systems are subject to component failures that tend to present major challenges to the control system.

1.2.1 Challenges of Controlling Large-Scale Complex Systems

In the control context, a control system controls a system to behave in a certain desired way to make the system accomplish one or more tasks, while satisfying constraints existing in the physical system states and control actions. The classic control strategy is feedback control, as shown in Figure 1.

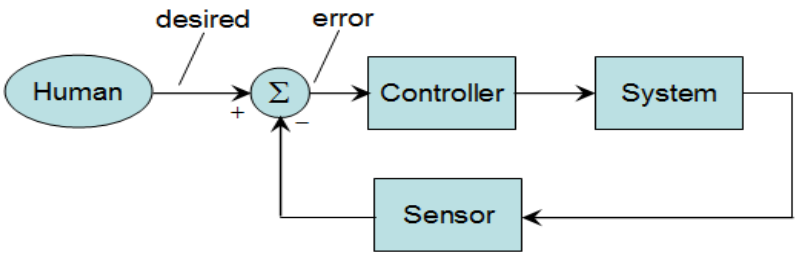


FIGURE 1 A CLASSIC FEEDBACK CONTROL DESIGN

There are two main types of feedback control systems: negative feedback control and positive feedback control. In a positive feedback control system, the set point and output

values are added. In a negative feedback control system, the set point and output values are subtracted. As a rule, negative feedback control systems are more stable than positive feedback control systems. Negative feedback also makes systems more immune to random variations in component values and inputs.

More generally, the control system has four steps: sensing, assessing, planning and action. First, the controller senses its environment and gathers the concerned information to prepare for assessment. Second, the controller analyzes the collected information and reasons about the system current states (such as detecting fault components, available resources and so on). Third, it decides control strategies which are objective-oriented based on an assessment of the current situation, without violating any system constraints. Objectives for a control system can be short-term goals, such as bringing the system to a desired state, or long-term goals, such as making the system achieve maximum rewards in a certain time period. Finally, the controller implements the control strategies in the real physical system. Such a general control sketch is illustrated in Figure 2.

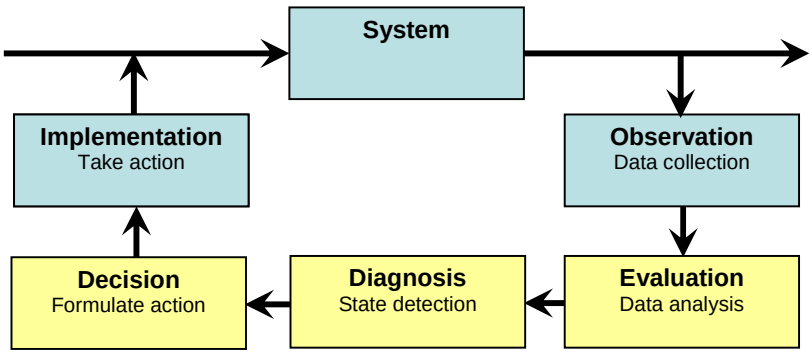


FIGURE 2 A GENERAL CONTROL SYSTEM DESIGN

Control of a large-scale complex system has to address the basic control issues mentioned above as well as a few more challenges:

- Uncertainty inference: relatively accurate and quick estimations of system states from noisy and incomplete information.
- Time delay: quick responses to local components' failures to avoid failure propagation.
- Robustness: automatic and optimal reconfiguration of the whole system, due to system dynamic changes/some failure modes without using predefined recipes.
- Flexibility: easy adaptation when parts of the complex system are modified or the system is extended.
- Global consistency: actions taken by local system and information gathered and processed locally need to be globally consistent.

1.2.2 Control Architecture

Carefully chosen control architecture is the prerequisite for a control system of a large-scale complex system being successful.

Classical automatic control systems based on feedback techniques and central control methods generally cannot manage computational complexity, nonlinearity, significant uncertainty and heterogeneity. Especially for the control methods that attempt to design point-design systems, it is hard to deal with situations that deviate considerably from nominal conditions and may lead to catastrophic results. Moreover, it is almost impossible to predict or list all possible damage scenarios and store the corresponding strategies in memory in order to reconfigure the system. More importantly, a modern control system must meet increasingly demanding requirements stemming from the need to cope with significant degrees of uncertainty, as well as more dynamic environments, to provide greater flexibility [9]. This, in turn, means that control system software is highly complex in that it invariably has a large number of interacting parts [8]. Furthermore, for a dynamical system with significant uncertainties, robustness of the control system is

highly desirable, especially for weapons and space systems. The usual way to accomplish this is to integrate extra copies of some critical components. This is called block redundancy. In contrast, a method that makes the system capable of achieving the same goal by using different methods with different components is called functional redundancy [10]. With either of these methods, the control system must determine the current configuration and states of the system and change to another configuration automatically. Identification of the system states is usually through the sensors on board. However, it is not practical and affordable to install sensors in every potential location of the system. The sensors as well as the components may fail over time; therefore, the observations from sensors may be incomplete and uncertain. Using one controller for the whole system as the central controller method does make the controller design extremely complicated and its reliability may be reduced considerably. Moreover, techniques for large-scale complex systems are being developed and constantly upgraded, which means more components and/or data have to be added and upgraded when these are made available. In order to increase the affordability of the control system, the control system needs to be easily modifiable, adaptable and expandable when parts of the system are changed or more data/components are integrated.

Conventionally, an automation system used for a large-scale complex system consists of sensors, analog filters connecting to sensors, actuators, cables, wires for transmission of analog quantities with very limited bandwidth, etc. The controllers operating online in real time are usually connected to the controlled actuators through relatively short-length cables/wires or optical fibres. Some physical devices need to handle issues such as signal distortion, noise interference and cable reliability which the controllers are designed to be robust to. Because of above-mentioned issues, the controllers controlling a distributed system are limited to a relatively narrow area. If there are a large number of entities in a complex system with multiple objectives in a widely distributed area with different time-

scales, it is very difficult to coordinate different part operations [11]. Conventionally, the control system designers try to decouple the system and do not coordinate the controllers, which limits the system's ability to accomplish more complicated tasks. However, for a highly coupled large-scale complex system, it is very difficult to achieve complete decoupling.

In summary, there are two general control architectures: centralized control and distributed control. Centralized control is a conventional control framework which has certain limitations for controlling large-scale complex systems. Distributed control as a modern control framework solves many problems exhibited in controlling large-scale complex systems. More detailed comparisons of these two control frameworks are given as follows.

1.2.2.1 Centralized Control

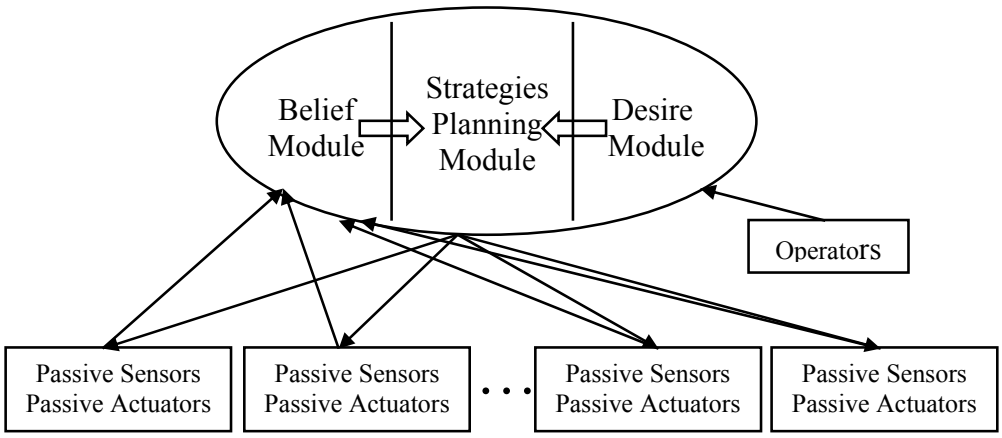


FIGURE 3 CENTRALIZED CONTROL FRAMEWORK

Centralized control is characterized as one particular designed part dominating the whole system absolutely. The dominant part acts as a master, while other parts act as servants. Usually, the master part is a complex processing unit which executes intensive

computation and communication. It actively collects information from its surrounding components and assigns tasks to its passive surrounding components. Figure 3 shows the structure of a centralized control framework. Although it is easier for centralized control to make the whole system work consistently and reach a global optimization point, it has high demands on communication capacity and computation power for the master entity. Most importantly, it takes significant time for the subsystem to deal with some local changes and it suffers from one point failure, i.e., if the master part is damaged, the whole system will be in chaos and out of control.

1.2.2.2 Distributed Control

In contrast to centralized control, a distributed control framework has more advantages. Distributed control can be further categorized into two types: conventional distributed processes and modern distributed processes.

In a conventional distributed process, the whole system is organized in such a way that the local data base of each processing node contains appropriate portions of the overall problem-solving data base needed by this local node's functions. This type of distributed process decomposition is called Completely Accurate, Nearly Autonomous (CA/NA) [12]. CA/NA distributed process is based on the assumption that each local node barely needs any assistance from other nodes in the system. It has all of the required information and capability to achieve its local goals, thus CA/NA is limited to those applications in which algorithms and control structures can be partitioned effectively so as to satisfy independency of each local node in the whole system. One way to provide independence of local processing nodes is to establish a global database which gathers all of the information of this system, and then provides access to each local node to acquire its needed information. In this situation, a CA/NA system is very expensive to implement due to the high communication and synchronization costs required to guarantee completeness and consistency of the local data and it is not really distributed either,

because the global database acts as an information processing center even though local nodes have the capability to perform certain functions based on the information they get from the global database. However, in this way, it does keep system state estimation consistent easily. Conventional distributed process can be viewed as an extended centralized control with distributed computation.

Modern distributed process is an alternative and new approach to structure distributed problem-solving systems for large-scale complex systems for which CA/NA is not suitable [12]. In this type of distributed process, the whole process is distributed into several local nodes and each node can achieve certain goals without knowing the whole system states. Figure 4 shows a modern distributed process in a completely parallel fashion.

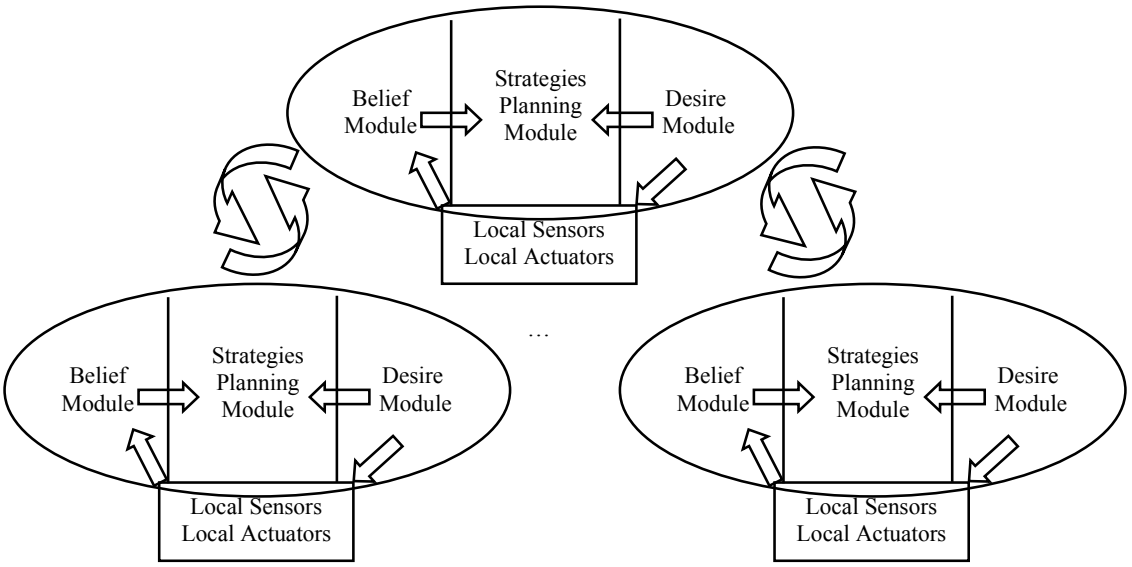


FIGURE 4 DISTRIBUTED CONTROL

In the processing time of a local node, it performs independently by using incomplete input data while simultaneously exchanging certain intermediate information of its processing with its neighbor nodes. By propagating appropriate information to its

neighbors, it can achieve a complete and global task with limited communication bandwidth for each local node. Furthermore, even before all of the nodes reach a global consistent consensus, each node can produce acceptable answers and take reasonable actions with incomplete, inconsistent intermediate results, which can avoid idle waiting under some damage scenarios, and make the system able to react to some local and imminent events. A system with this type of problem-solving structure is called Functionally Accurate and Cooperative (FA/C) [12]. By using a modern distributed process, the system can work robustly under some node's failures since each node can work appropriately under the situation with missing information. If there is no failure of any local node and the communication works in a perfect way, such a FA/C distributed system can attain the same results as a centralized controller, but it responds more quickly with less communication and cheaper local processors. In summary, the advantages of a FA/C distributed system are listed as follows [13]:

- Increased reliability and flexibility---avoid single-point failures and achieve more reliability through redundancy in communication paths and cheaper processors.
- Enhanced real-time responses---local nodes can work in a parallel fashion. Controllers are installed close to the real systems and can respond quickly with very short communication delays to some local imminent events.
- Lower communication costs---not all of the information for a local node is transmitted to its neighbors. It transfers a relatively small portion of its local information at an abstract level to its corresponding acquiring neighbors.
- Lower processing costs---the whole processing is distributed to a set of local nodes, which dramatically reduces the computation power requirement for individual processors. This means for local control processes, cheap and conventional processors can be used. Normally, a set of lower powerful processors are cheaper than a super powerful processor.

- Reduced controller complexity---achieved by decomposing a whole problem-solving task into a set of subtasks. Each subtask focuses on a smaller specific domain; the experts for such a domain can produce a more effective and better design.

Distributed control is the trend in controlling large-scale complex systems. The following example of navy ship control system evolution over time sheds a light on such a trend.

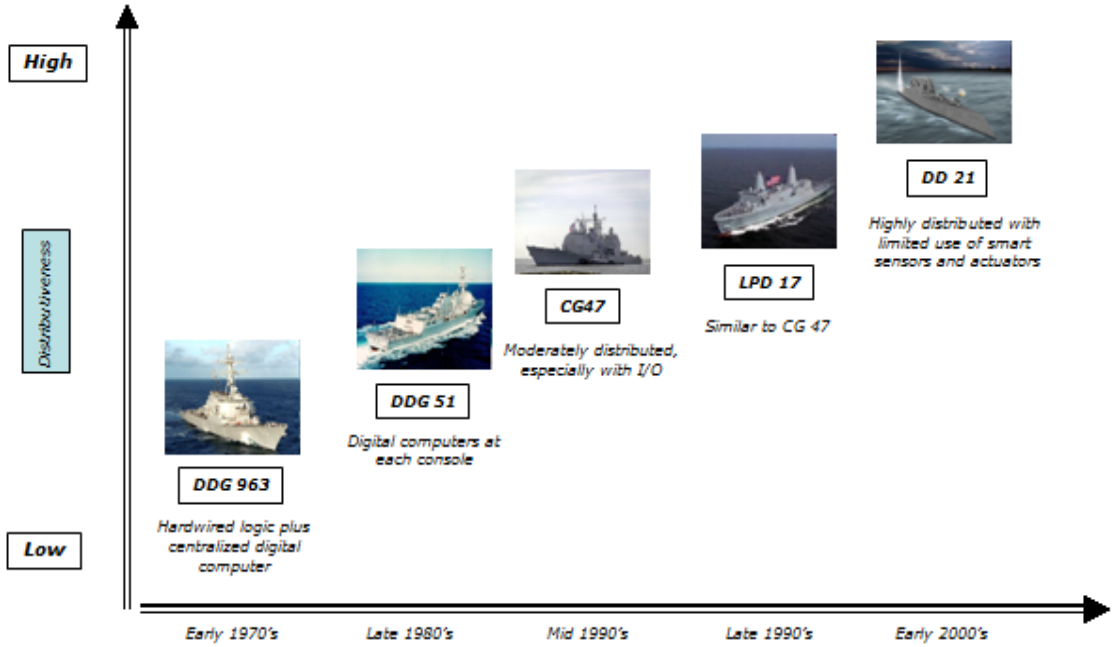


FIGURE 5 NAVY SHIP CONTROL SYSTEM EVOLUTION

Early automated systems on navy ships had the user interface, processing electronics, signal conversion electronics, and the network interface contained within a single console. With the development of computing and network technology, a portion of the electronics was moved to "satellite" enclosures which were connected through a network. The earliest example implementing this technology in the US Navy is the USS Yorktown (CG48) which formed the basis for smart ship installation. In the CG48 monitoring and control system, the signal conversion electronics were moved from the main engine room console to several remote units that were networked together, while the operator interface,

network interface, and the "intelligence" were still located in the main engine room console. Later on, all the computing and signal/data conversation functions were separated from the main engine room and distributed to local computing units [14]. Currently, the automation systems of US Navy ships are highly distributed with limited smart sensors and actuators. Figure 5 shows the development history of US Navy ship automation systems. How to make such automation systems more stable, more efficient and more intelligent in significantly uncertain and dynamic environments is still an on-going research field. This is also the objective of the research in this dissertation.

Distributed control system with high intelligence is the trend of controlling large-scale complex systems. However, there are challenges as well when compared to centralized control and conventional distributed processes as described as follows:

- How should the whole system be decomposed into a set of subsystems? Should it be decomposed spatially or functionally or by using both simultaneously? Is there a general decomposition method for large-scale complex systems or it is problem-specific?
- What level should the decomposition process reach? What size of subsystem is more reasonable and more effective? Too few subsystems would not display the advantages of distributed control while too many subsystems would be trivial and make the coordination more complicated. How should we achieve the balance?
- How should an individual subsystem to achieve its goals relatively independently but still contribute to the global tasks?
- What kind of information needs to be transmitted between a node and its neighbors? How should we abstract the information? Is there a standardized way for the communication among heterogeneous entities?

- How to make the knowledge located in local nodes globally consistent by just communicating between neighbor nodes?
- How to solve the conflicts among different objectives from various entities and achieve a global task coordinately?

In the next section, we will give a simple introduction to a few available techniques which support distributed control for large-scale complex systems.

1.3 Techniques Supporting Distributed Control

For a modern control system, there are many available techniques which support distributed control of highly coupled, multi-objectives and widely distributed complex systems consisting of numerous heterogeneous entities with different time-scales. For example, "*recent advancements in micro processor based protective relaying and other intelligent devices that can be integrated into auxiliary electrical systems, coupled with advancements in plant control system are the driving force behind the transfer from traditional hard-wired control schemes to the control of auxiliary electrical systems via communication networks*" [15].

Here, a few existing and general techniques are listed as follows:

- Powerful processors with small physical size can handle very complicated control algorithms in a digital domain in a very short time. It can be shown that digital controllers can provide at least the same results as analog controllers. For example, the operation of analog filtering followed by sampling can be performed if one chooses to sample in a certain frequency firstly and then operate in the sample (digital) domain [16]. The theorem supporting this statement is shown below:

Theorem: Let $x(t)$ and $h(t)$ be stable continuous signals, based-band (B), where $\sum_{n\in Z}\left|x\left(\frac{n}{2B}\right)\right|<\infty$, $\sum_{n\in Z}\left|h\left(\frac{n}{2B}\right)\right|<\infty$ and $y(t)=\int_R h(t-s)x(s)ds$, then

$$y\left(\frac{n}{2B}\right)=\frac{1}{2B}\sum_{n\in Z}h\left(\frac{n}{2B}\right)x\left(\frac{n}{2B}\right) \tag{1.1}$$

This is the theorem of the equivalence of analog and digital filtering (proof see [16]).

Furthermore, there are more advantages of digital processor control:

- By using a small size processor, it is much easier and cheaper to configure and reconfigure a control system via software.
- Digital processors are much less sensitive to environmental conditions than some analog devices, such as capacitors, inductors and cables.
- Digital processors are more scalable in handling a complex task and can decompose a complicated task into several sub-tasks in a more flexible way.
- By using digital processors, it is easy to implement some control algorithms with dynamically changing parameters (formally, it is easy to implement adaptive control methods).
- Digital processors are much cheaper than analog controllers.
- High speed communication networks with reliable error detection and automatic correction can support real-time control in a large-scale complex system. Rapid advances in wireless packet networking technology make it possible that large quantities of data can be transferred reliably and efficiently though wireless networks in a highly distributed system with relatively low cost. Sensory data coming from multiple sensors of different modalities in distributed locations are used to implement

the concept called smart environment, which represents the next evolutionary development step in building, utilities, industrial, home, shipboard, transportation systems automation, etc [17].

Numerous techniques supporting distributed control for large-scale complex systems exist. Listing all of them would be impossible and beyond the scope of this dissertation. In this dissertation, we will assume those techniques are available and focus on the methodologies and processes of designing a control system with inference engine for a general large-scale complex system.

Furthermore, various development environments supporting distributed control system designs are available to shorten the design time of distributed control systems. Such development environments are discussed in Chapter II.

1.4 Research Objective

The general research objective of this dissertation is to ***develop a generalized and comprehensive framework for the control system design of a general large-scale complex system under significant uncertainties***. This general research objective has been decomposed into three sub research objectives to make it clearer and more understandable as follows:

- ***O1: Establish a general control architecture for large-scale complex systems that can provide the capability of being robust, flexible, reusable, and scalable.***
- ***O2: Establish an inference engine that can handle incomplete and noisy information, and make inference reasoning automatically, consistently, efficiently and robustly. This inference engine should have the capability of reaching global state consistency under some conditions and at the same time, keeping privacy of***

subsystem domains, i.e., each subsystem revealing partial information to the public or its concerned neighbors.

- ***O3: Integrate the inference engine into the control architecture to make them work smoothly.***

1.5 Dissertation Organization

In Chapter I, an introduction to the background of this research is given. Chapter I also shows what the challenges and prerequisites for this research are. The state-of-art general methodologies for controlling large-scale complex systems are investigated. It also lists what problems this research is trying to solve and what aspects this dissertation focuses on.

Chapter II focuses on establishing general control architectures for controlling large-scale complex systems. A new hybrid control architecture with distributed multiple agents will be proposed. Detailed comparisons of the new control architecture with existing control architectures will be discussed quantitatively. Furthermore, different distributed control system's design environment/software will be introduced. Java Agent Development (JADE) framework as an open-source and flexible multi-agent system design environment will be chosen as the tool for the implementation of the methodology and process proposed in this dissertation. More detailed information about JADE will be shown in Chapter II.

Chapter III will give a simple review of preliminary knowledge in the probabilistic inference field. It helps the reader fully understand what the proposed methodology and process are in this dissertation. It makes this dissertation complete and self-contained.

Chapter IV will focus on the distributed inference engine and how to embed such a distributed inference engine into the new designed hybrid control architecture. A detailed discussion of different inference engines also will be given. Through the comparisons, Multiple Sectioned Dynamic Bayesian Networks (MSDBNs) will be chosen as the distributed inference engine and more detailed information on MSDBNs will be given. However, in order to make globally consistent inferences, three requirements: tree structure, d-sep set property, and running intersection property need to be satisfied before an MSDBNs makes inferences. Detailed discussion about automatic satisfactions of the three requirements in a distributed way will be given and three corresponding algorithms will be proposed.

Chapter V will summarize the proposed methodology and process in this research and help the readers refresh their minds and get to the core of this research quickly.

Chapter VI is the application part of this dissertation. Control system design of a simplified ship Chilled Water System (CWS) of DDG51 as an application of the proposed methodology and process will be implemented. Detailed introduction of DDG51 ship CWS will be given. A step by step procedure for designing the control system using the proposed methodology and process will be described in detail.

Finally, Chapter VII summarizes the lessons learned from developing and implementing the proposed methodology and process. It will also list the main contributions of this dissertation and identify areas related to this research field as a guideline for future work.

CHAPTER II

MULTI-AGENT BASED CONTROL ARCHITECTURE

"The smartest people in the world do not generally look very intelligent when you give them a problem that is outside the domain of their vast experience."

-Herbert Simon

As mentioned before, a complex system consists of many heterogeneous components. The control system design for a complex system is a difficult project which needs numerous people to work coordinately over a significant time span. One designer knows one domain very well, but may know little about another domain. Furthermore, in most situations, the designers want to keep their domain knowledge private and are only willing to provide part of the internal structure and data to the public or to other specific entities. Since the 1970s, the Multi-Agent Based Control (MABC) method has been the focus as a possible solution to the solve above-mentioned problems in controlling a large-scale complex system.

2.1 Agent Introduction

A system consists of many agents. The first question which comes to mind is what is an agent. The definition of an agent is not unique. Different people with different applications may give different descriptions. In this dissertation, the definition of agent is prone to the context of control systems. Basically, an agent has a general functional architecture. It takes in sensor data as well as data from other agents; it provides data to its neighboring agents as well as commands to actuators. Internally, a decision making

module processes sensor information and incoming messages, and issues messages to the rest of the system [18]. Each agent has a clear interface and boundary in interactions with other agents, such as what inputs it needs and what outputs it offers. Each agent has its own logic to decide the behaviors of itself according to its environment which is determined by its inputs. Each agent affects the other agents' behaviors by its outputs. Note that inputs are not necessary from the sensors and the outputs are not necessary to the actuators. In summary, an agent has the following characteristics as claimed by Wooldridge and Jennings [9]:

- Clearly identifiable problem-solving entities with well-defined boundaries and interfaces.
- Situated (embedded) in a particular environment over which they have partial control and observability: they receive inputs related to the states of their environment through sensors and they act on the environment through effectors.
- Designed to fulfill a specific role: they have particular objectives to achieve.
- Autonomous: they have control over both their internal states and their own behaviors.
- Capable of exhibiting flexible problem-solving behaviors in pursuit of their design objectives, being both reactive (able to respond in a timely fashion to changes that occur in their environment) and proactive (able to opportunistically adopt goals and take initiatives).

According to the above definition, an agent could be a mechanical system, a person, a smart dog, a piece of software, etc. For this dissertation, an agent most likely refers to a piece of software in which some control algorithms are embedded, used as an intelligent controller. It is different from traditional mechanical agents or a real human being. A software agent in the digital domain should at least reach the same goal as a traditional

mechanical agent does. Actually, a software agent can implement more complicated algorithms in a clearer, easier and more adaptive way.

2.2 Agent Structure in Control Domain

Control agents are autonomous with some level of intelligence, which does not mean that a human cannot control or have certain effects on the agents. Intelligent agents can be responsive to commands or requirements and react in a certain way to meet certain objectives the human or operator are trying to attain. However, it does not need a human to instruct it how to act step by step. An agent works in a more efficient, robust and autonomous way. Particularly, for an intelligent control agent, it has its own thinking logic, described by Scheidt as a cyclic process designed to accomplish the desired behavior of the system being controlled by this agent. The control cycle consists of three steps: estimation, planning and execution [19]:

- In the step of estimation, an agent perceives information from its surroundings (other peer agents, child agents or parent agents or operators) and its own local sensors. All of the information is raw data. The agents have the capability to process the data, such as filtering out noise, distinguishing wrong data, recognizing some states patterns and organizing the data into a more useful and abstract format, and so on. By processing the raw data, the agents form their own beliefs about their environment and internal states. These organized and relatively accurate data are used by the agent to make strategies to satisfy its own objectives that are related to the global objectives of the whole system.
- The second step is planning. The agent uses its belief database to plan a set of control strategies to maximize the control objectives within state and control constraints.

- The last step is execution. The agent translates the control plans into real signals which are acceptable to physical actuators.

This cyclic process of an intelligent agent is illustrated in Figure 6.

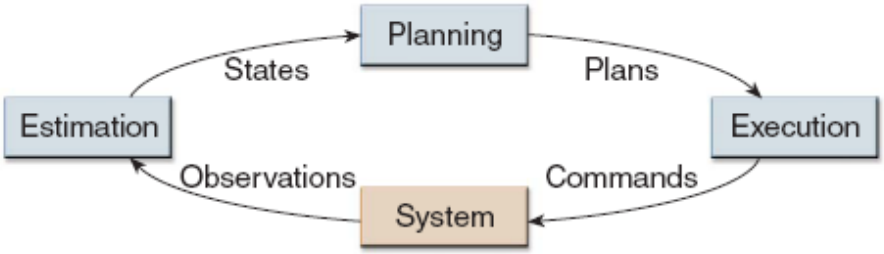


FIGURE 6 A CYCLE PROCESS OF AN CONTROL AGENT [19]

This concept of control agent cycle process is very similar with the idea of sense, assess, control and actuate [14]. A more widely accepted and implemented logic thinking of an agent is Belief-Desire-Intention (BDI) [7]. A BDI agent is a particular type of bounded rational software agent, imbued with particular mental attitudes: Beliefs, Desires and Intentions. BDI is very similar to the above three steps: estimation, planning and execution. Beliefs are the agent's knowledge base which specifies the agent environment and its internal states from the perspectives of its own eyes. Those beliefs should be updated appropriately after each sensing action or message reception. Desires are the various objectives which the particular agent wants to achieve at the current time. It can be regarded as representing the motivational state of the system. Intention is the actions that the agent can commit to satisfy its desires based on its current belief and system constraints. BDI architecture was originally developed in the early 1990s, based on a model of human reasoning developed by Bratman [20]. Procedural Reasoning System (PRS), based on BDI framework for intelligent agents shown in Figure 7, is a popular agent architecture in artificial intelligence systems. Typically, the belief, desire and

intention components are designed and embedded explicitly in the memory of a PRS agent at runtime and can change dynamically, which distinguishes from purely reactive systems.

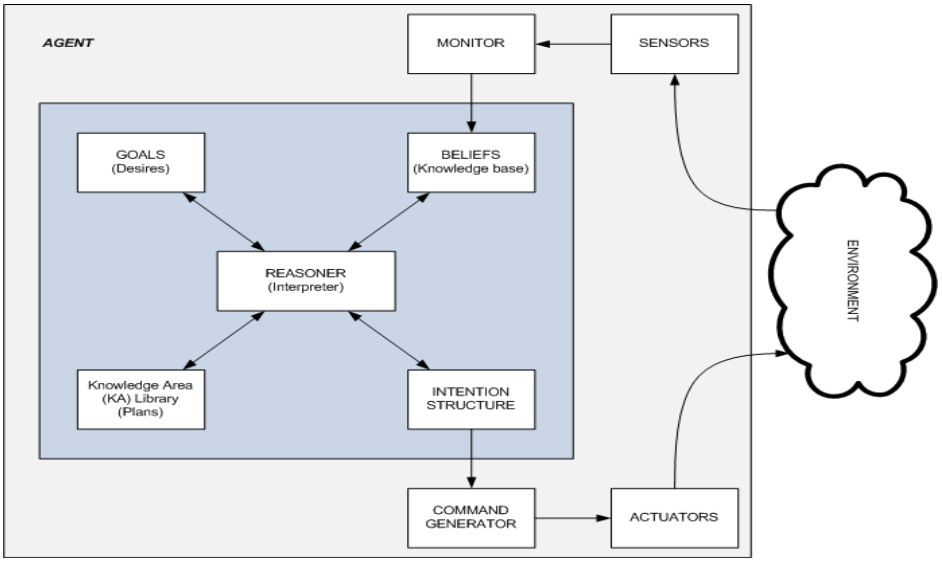


FIGURE 7 PROCEDURAL REASONING SYSTEM (PRS) [21]

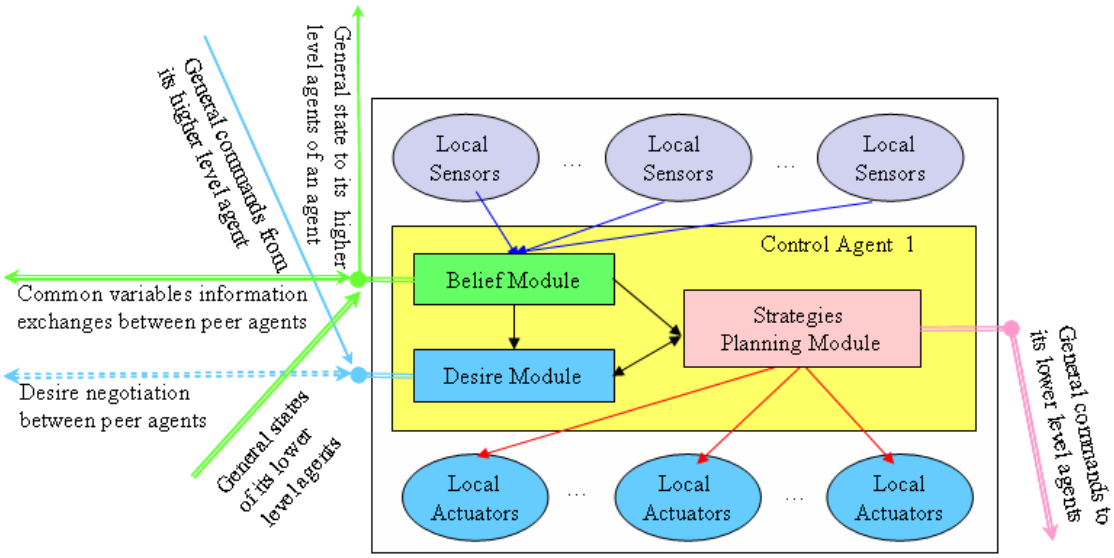


FIGURE 8 GENERAL INDIVIDUAL AGENT STRUCTURE

Considering that an agent is working in coordination with other agents in an integrated system to achieve a global task, a general structure extending from BDI agent structure can be shown in Figure 8. In Figure 8, an agent's core brain has three parts: belief module, desire module and strategy planning module. Every module has a specific function and gathers/distributes different information. The belief module gathers information from its local sensors as well as its neighbor agents. If a hierarchical architecture is used for the whole system, the belief module gets information from its lower level agents as well. The belief module has the capability to process the raw data from its sensors, such as filtering out noises, classifying data types, distinguishing wrong and damaged information, etc. More importantly, the belief module has reasoning capability, i.e., it can use incomplete and noisy information to infer some hidden variable states which are required for the agent's planning. Furthermore, the belief module has the capability of compiling the detailed information for this local agent into an abstract style and automatically exchanging messages about state information with other agents in a more concise way. In summary, in order to resolve noisy and incomplete information, an agent must have the capability of detecting inconsistencies from its local sensor information and the messages it receives from other agents on some common variables. Basically, the variables in a specific agent can be categorized into two types: completely local variables and shared variables with other agents. The two parts of variables are connected in some way. For the completely local variables, it accepts whatever reasoning results from the local information. However, for the shared variables between two agents, the reasoning module needs to have the capability to resolve conflicting conclusions about their states by using some iterative procedure to make these two agents converge to consensus, drawing out all of the relevant information existing in both of the agents. The desire module forms its control objectives based on its current environment and internal states and commands from the operators. The strategies and planning module establishes control actions according to the agent's states and its control objectives. Additionally, in the structure

shown in Figure 8, the communication module is collapsed in order to make a clear view. For a real application of an agent, the communication module is an inevitable part which is responsible for the agent passing/receiving messages from other agents or real physical systems. What communication protocols are used and how to identify senders' identities in multiple received messages, or how to wrap the load/envelope in one message is beyond the scope of this dissertation and will not be discussed further. In this dissertation, we assume the communication techniques are ready for use, because many available agent development environments support different communication techniques without users' efforts of designing it from scratch. Such ready-to-use software development environments will be introduced later.

2.3 Introduction of Multi-Agent Based Control

We have already discussed how an individual agent works in the previous sections. For controlling a large-scale complex system, it always requires numerous agents to work cooperatively to achieve one or more global control goals, which is called a Multi-Agent Based Control (MABC) system. Basically, a MABC system is implemented by decomposing a complex system into many small parts; each part is treated as a relatively isolated agent controlling a local region intelligently and interacting with other parts of the whole system.

2.3.1 Advantages of a MABC System

A MABC system has several advantages for controlling large-scale complex systems as an implementation of the modern distributed control framework as discussed in Chapter I:

- It divides a big task into smaller and more manageable pieces, which makes it easier to implement many modern control methods such as neural nets control, fuzzy logic control, etc.

- It separates the controller from the physical system. The controller is designed as software that can handle computational complexity.
- It makes the system more robust to uncertain and dynamic environments.
- It makes the system more flexible and adaptable as time passes, especially when agent designs for some similar problems are standardized.

However, agent-based control is not a panacea, and gains in complexity management and robustness can be offset by decreases in effectiveness and ability to prove correctness [22]. Careful control architecture design and individual agent internal logic design are always necessary to make the control system work in a more effective and robust way.

2.3.2 Challenges of MABC System Design

A multi-agent-based control system includes many agents that are working interactively. For a simple agent, it is easy to predict the responses under certain situations. However, it is very difficult to theoretically evaluate the integrated effects of a multi-agent-based control system, whose control behavior emerges from the concerted activities of many agents. The autonomy of the agents and the fact that interactions occur for unforeseeable reasons leads to the unpredictability of the overall behaviors of the runtime system [23]. Extreme situations may cause the whole system to crash due to logical errors in the interactions among agents, thus testing and evaluating the agent-based control systems is as important as designing the controllers. Testing the control system is an iterative process; therefore the use of the real physical system to test the control system is highly costly and ultimately unrealistic. Using a simplified simulation model instead of a real physical system is the most effective way to do a preliminary check of the designed MABC system.

For a MABC system, the controllers are separated from the physical system. They are connected to each other by inputs and outputs. The inputs of the control system are from

the sensors of the physical system and the outputs are given to the actuators of the physical system. Experts in designing agents in a specific area can test their own parts first, and then iterate with designers of other area agents through the interface. However, MABC, as an implementation of modern distributed control framework, has to face a few challenges as a modern distributed control framework does. We restate those challenges as follows:

- How should the whole system be decomposed into a set of subsystems? Should it be decomposed spatially or functionally or by using both simultaneously? Is there a general decomposition method for large scale complex systems, or it is problem-specific?
- What level should the decomposition process reach? What size of a subsystem is more reasonable and more effective? Too few subsystems will not display the advantages of distributed control while too many subsystems will be trivial and make coordination more complicated. How to achieve the balance?
- How should an individual subsystem achieve its goals independently and still contribute to the global task?
- What kind of information needs to be transmitted between a node and its neighbors? How to abstract the information? Is there a standardized way for the communication among heterogeneous entities?
- How to make the knowledge located in local nodes globally consistent by just communicating between neighbor nodes?
- How to solve the conflicts among the different objectives of various entities and accomplish a global task in a coordinated manner?

Finding a perfect solution to all of the questions listed above is a very challenging task. In the following context of this dissertation, the author will try to narrow down the scope of

this research and focus on three specific aspects originated from the general research objective addressed in Chapter I:

- ***O1: Establish a general control architecture for large-scale complex systems that can provide the capability of being robust, flexible, reusable and scalable.***

- ***O2: Establish an inference engine that can handle incomplete and noisy information, and make inference reasoning automatically, efficiently and robustly. This inference engine should have the capability of reaching global state consistency under some conditions and at the same time, keeping privacies of subsystems' domains, i.e., each subsystem revealing partial information to the public or its concerned neighbors.***

- ***O3: Integrate the inference engine into the control architecture to make them work smoothly.***

Corresponding to the first research aspect of this dissertation, the first research question is arising:

- ***Q1: Is there a distributed control architecture that can provide the capabilities of being robust, scalable, flexible and reusable for controlling large-scale complex systems with limited communications?***

In the rest of this chapter, different multi-agent based control architectures will be compared and a new architecture combining the existing multi-agent based control architectures will be proposed.

2.4 Multi-Agent Based Control Architecture

2.4.1 System Level Requirements

In order to solve the issues existing in MABC systems, an effective and efficient MABC architecture needs to be established to provide four qualities: ***robustness, flexibility, reusability and scalability.***

- ***Robustness*** implies that the control system can reconfigure the system based on the current system states and objectives, such as detecting faults, isolating faults, invoking an alternative to damaged parts, compensating for disturbances, etc. The control system needs to have the capability to reconfigure the system in an intelligent and automatic way. For example, assume there are several routes for a service load to get chilled water resources from two alternative resource centers. When the current route is damaged, the agent in charge of the connection of the service load with the resource center needs to be aware of this situation and try to switch to another route first by manipulating relevant valves, instead of shutting down the service load or switching to another resource center since frequently shutting down or turning on a resource center will shorten the pump/chiller lifetimes.
- ***Flexibility*** is the ability of the control system to change its control parameters or objectives at different times according to the system requirements and the global objectives of the control system or some operators' inputs.
- ***Reusability*** is the ability of an existing control system to be used for a new system with slight modifications. A large-scale complex system is always extended over time. Redoing the whole system due to extension is expensive. If some existing models can be reused when more components need to added to the system, it will save a lot of money and time. For example, the rate of change of automotive products is accelerating, and it is increasingly important to reuse equipment to produce

different models. It requires the ability to change both the functions of a machine and its interfaces. These interfaces operate at two levels: control and mechanical. Ideally, interfaces should be standardized to permit the factory to plug-and-play machines in the line. This ability is greatly enhanced if machine agents can recognize the environment in which they are being installed and modify themselves accordingly [24].

- ***Scalability*** of a control system is very important to large-scale complex systems. It means that the control system can be easily adapted when more components/data are added to the system. In order to use some new techniques or add some new functions for a complex system, if the system is not scalable, it may be necessary to redesign the whole control system completely, which would be very costly. By using agent-based control, adding new techniques may just require changing some internal algorithms of one particular agent and extending the interfaces for several agents, or just pluging a new agent with interfaces to other agents.

2.4.2 Literature Review of Distributed MABC Architectures

Karray proposed a distributed control and coordination of multi-agent systems framework [25] based on the idea of Hybrid Control Intelligent Agent (HCIA) discussed by Fregene [26]. In this framework, each agent is modeled as a Coordinated Hybrid Agent (CHA) composed of an intelligent coordination control layer and a hybrid control layer. The intelligent coordination control layer deals with the planning, coordination, decision-making and computation of the agent. The hybrid control layer takes the output of the intelligent coordination layer and generates discrete and continuous control signals to the physical system. Two agents coordinate with each other by initial constraints or through direct communications to add more constraints. Those constraints are more like rule-based conditions. The agents do not provide the capability to learn and evolve with the operation process. Each agent will try to attain an optimal action sequence under the

trajectory and action constraints. However, this paper does not discuss the stability and robustness of such a framework. It could not guarantee global information consistency under an environment with uncertain and incomplete information. Parunak gave a review of industrial agent applications (focusing on manufacturing lines) from a practioners' perspectives [24]. He categorized agent-based applications for manufacturing lines into four areas: manufacturing scheduling, control, design collaboration, and agent simulation. In the control part, he discussed how agents can contribute to the goals of robustness, flexibility, reusability, and scalability in a more direct and effective ways than non-agent approaches. An electrical power network is a widely researched area for agent-based control application. Wu justified that the technical developments in fast data communication network technology opens up the possibility of parallel and distributed implementations of the state estimation algorithms in an electrical power network system [27]. A parallel and distributed processing in state estimation of power system energy had been discussed by Carvalho and Barbosa [28]. The power network is decomposed into K areas which are connected through boundary buses belonging to both adjacent areas simultaneously. Each area has its own processor and uses the Weighted Least Square (WLS) method to estimate local actual states from local observations with noise. In order to satisfy the boundary conditions, each processor updates its boundary area states according to its neighbors' states in an iterative manner. However, the consistency of the whole system about the boundary conditions is not justified. The author mentioned that such discrepancies in values of boundary bus state variables estimated by using different sets of measurements could be acceptable and the effect on computation efficiency could be minimal if higher redundancy levels were used in the whole system, but no quantitative results were shown in the paper. Incomplete and bad data problems are not discussed in this paper either. In Carvalho and Barbosa's later publication, a distributed bad data detection and identification schema by using statistical hypothesis test with fixed thresholds is discussed [29]. However, neither experiments nor further discussion about

the accuracy and efficiency of such a schema is provided. Romanovski and Caines extended from the central classical supervisory control results for scalar systems to the more general Multiple Agents (MA) product system cases and proved that there exists an algorithmic procedure for the recursive construction of an MA supervisor when additional subsystems/components are added to a system via the MA product [30]. However, this framework focuses on this particular task of detecting and combining new automation to the MA system and it does not deal with uncertainty and multiple agents' coordination. Naso and Turchiano proposed a distributed multi-agent approach for dynamic part routing in automated manufacturing systems [31]. The control strategies in this paper allow the part agents to make decisions not only on the imminent operation, but also on subsequent ones. The anticipated decisions from different part agents are transmitted to workstation agents that will scrutinize those decisions and resolve conflicts by modifying part agents' decisions. In order to resolve conflicts, the workstation agents need to gather part agents' state information. It may take more time but does not make much difference from the method that the workstations gather state information from part agents and assign doable tasks to each part agent. And this method is designed especially for manufacturing systems and not easy to be extended to other implementations.

To the best knowledge of the author, almost all of the agent-based control architectures for large-scale complex systems under significant uncertainties are domain specific and not easy to be extended to other domains. This situation is partly due to the fact that control architecture of multiple agents is determined by agent to agent coupling and interaction mechanisms. How different agents couple and interact with each other depends not only on the way that the whole system is decomposed but also on the specific systems. However, generalized agent-based control architecture can still be established by choosing an appropriate decomposition scheme and representing relationships between different parts in an abstract and standardized way.

2.4.3 Existing MABC Architectures

As described before, many types of architecture of agent based control for various applications have been proposed in the past decade. However, they can be categorized into three main independent types: flat architecture, hierarchical architecture and module architecture.

2.4.3.1 Flat Architecture

The first type is flat architecture in which all of the agents are in parallel. Any agent can communicate with any other agents directly. It dramatically increases the autonomy of the individual agent and makes the control system more flexible and adaptable. However, each agent needs to be registered with specific functions (services), and the system needs to provide an efficient search algorithm to search all of the services. It is very difficult to deal with objective conflicts and predict emergent situations considering all of the agents intervening with each other, which makes the system fragile to some unexpected emergent situations. More importantly, it is extremely hard to design an algorithm which can be used efficiently and will converge on consensus of individual agents for a large scale complex system with multiple global objectives and intensive interactions among different part agents. Flat architecture can be further divided into two types, full grid communication and common bus communication as shown in Figure 9. Full grid communication is based on point-to-point communication, thus it is more reliable but more complicated. Common bus communication structure is to use a shared bus crossing the whole system. All of the agents send and receive information from this shared bus.

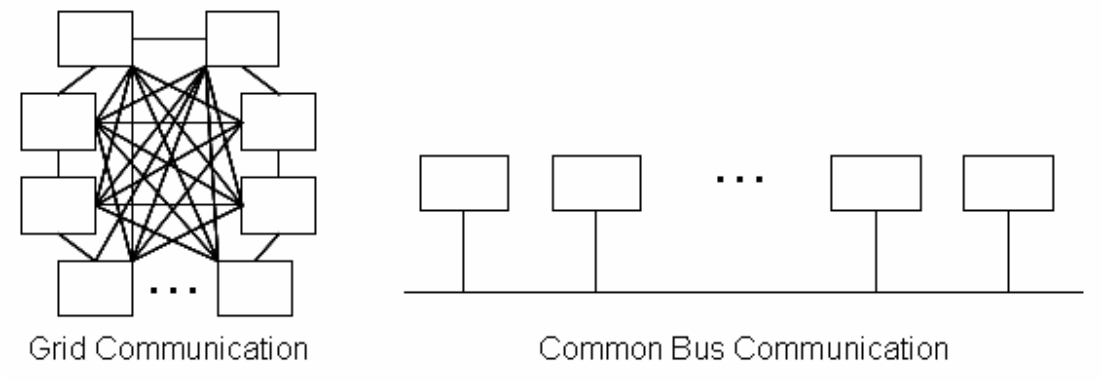


FIGURE 9 FLAT ARCHITECTURES

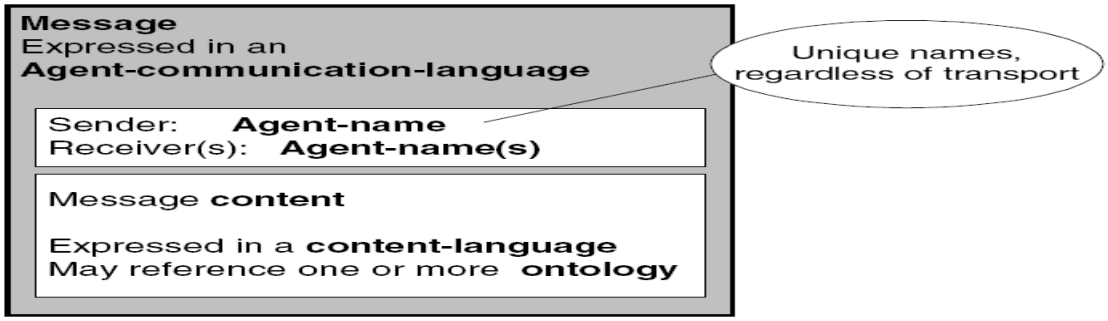


FIGURE 10 A MESSAGE STRUCTURE [32]

As shown in Figure 10, messages transferred between agents have two parts: envelope and payload. The envelope includes the sender and receiver information as well as how to transfer the message. Payload is the main content of a message which is encoded by using the encoding-representation appropriate for the transport. Therefore, even though all of the messages are through the same bus, they will be delivered to the desired destinations. However, as we can see from Figure 9, if there is a break point in the common bus, the whole agent set is divided into two independent sets between which there is no message passing.

2.4.3.2 Hierarchical Architecture

The second architecture is hierarchical architecture. It is a natural way to decompose a system hierarchically. This architecture reduces the design complexity of individual agent and communication networks. It provides effective control of complex systems that have multiple hierarchical goals, multiple sensors and a need for robustness. More specifically, Jennings and Bussmann argued that there were five reasons for using hierarchical multi-agent-based control for a complex system, based on the characteristics of the system itself [9]:

- The relationships between agents are made much clearer and easier to understand.
- Complexity frequently takes the form of a hierarchy.
- Which components in the system are the primitives is a relatively arbitrary choice and defined by design objectives.
- Hierarchical systems evolve more quickly than non-hierarchical ones of comparable size.
- Using this approach, it is possible to distinguish between the interactions among subsystems and those within subsystems.

However, for hierarchical architecture, lower level agents depend on higher level agents. Once the higher level agent has some problems, lower level agents connecting to it will lose supervision. Although the lower level agents can work in a default way without inputs from higher level agents, it will lose the ability to keep itself aware of the whole system and isolate itself to just do the local control, which may cause system crash in a large-scale complex system.

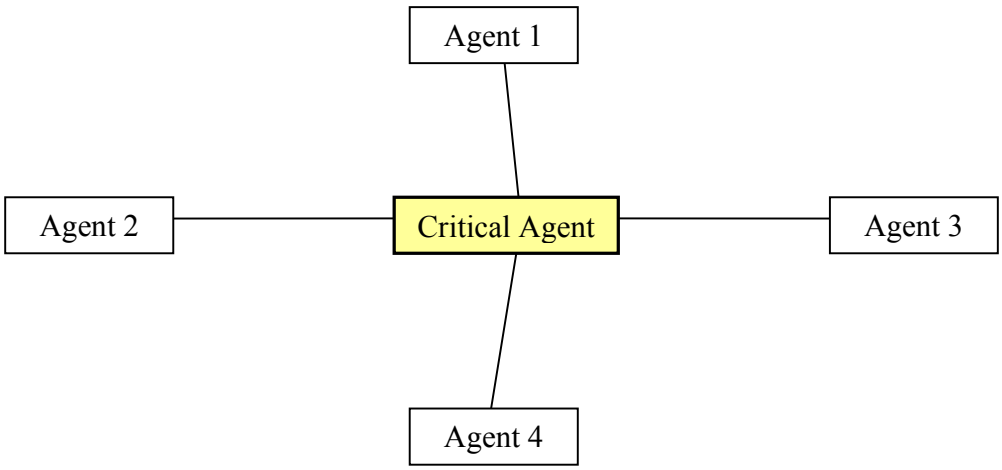


FIGURE 11 A CRITICAL AGENT WITH FOUR SUBLEVEL AGENTS

For example, an agent connecting with four sublevel agents has the capability to make plans for those four connected agents' actions and report abstract information from these four agents to a higher level agent as shown in Figure 11. If the critical agent fails, the four lower level agents lose guidance and become orphans, without knowledge of the states of other agents in the whole system. Based on that, some special logic and coordination need to be done to decrease the dominance of higher level agents and increase the autonomy of lower level agents. There are various architectures inherited from hierarchical architectures. For example, Scheidt implemented a hierarchical control for ship auxiliary systems (currently just for the ship wide chilled water system) as shown in Figure 12 [19]. Open Autonomy Kernel (OAK) is a distributed agent control implemented by the Johns Hopkins Applied Physics Laboratory (APL). It is a "high level" three step control process: diagnosis, planning and execution. OAK is specifically designed to support difficult control programs for complex systems with incomplete sensor data with distributed control [33]. However, this architecture does not support a globally consistent inference engine in distributed ways.

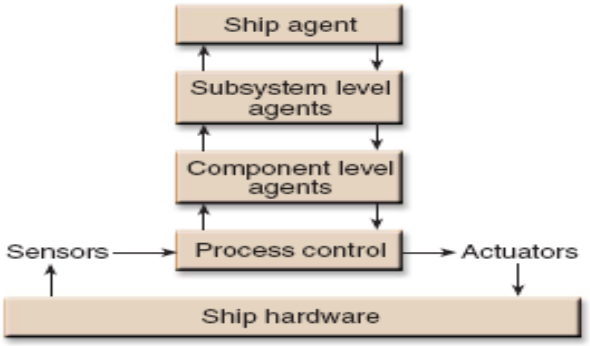


FIGURE 12 THE HIERARCHICAL CONTROL ARCHITECTURE IMPLEMENTED IN THE OPEN AUTONOMY KERNEL [19]

2.4.3.3 Modular Architecture

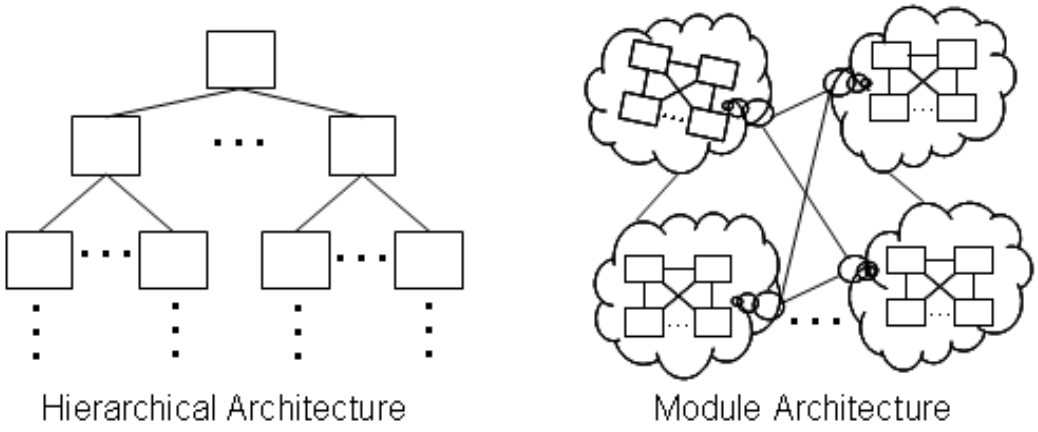


FIGURE 13 HIERARCHICAL ARCHITECTURE AND MODULE ARCHITECTURE

The last architecture is modular architecture which divides the whole system into several modules. Each module uses flat architecture in which agents can communicate with each other directly. However, the agent in one module communicating with agents in another module goes through the modular common interfaces. Modular architecture reduces the design complexity of the communication system and at the same time it keeps individual agent autonomy to some extent, therefore, it is the appropriate choice for most applications. However, there is no general approach to how to decompose a general

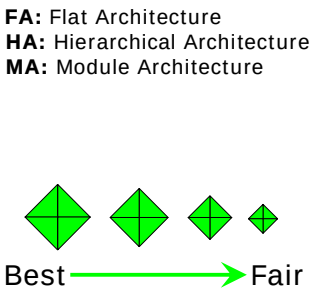
large-scale complex system into several modules. Too many modules may face the same problems as flat control architecture does. Hierarchical architecture and module architecture are shown in Figure 13.

From the previous discussions, it is clear that each of the existing multiple agent-based control architectures has its own advantages and disadvantages. In Table 1, the three existing control architectures are qualitatively compared from six perspectives: structure complexity, communication complexity, coordination complexity, autonomy, reliability and adaptability.

Flat control architecture has good autonomy, however, it suffers from structure complexity, communication complexity, coordination complexity and poor adaptability. Hierarchical control architecture is excellent except for its low autonomy and reliability. Module control architecture is acceptable in most situations. However, it still has much room to improve, such as communication among different modules, handling structure complexity, system level convergence, etc.

TABLE 1 EXISTED MULTIPLE AGENT-BASED CONTROL ARCHITECTURES

TYPE	FA (Common Bus)	FA (Grid)	HA	MA
Structure Complexity				
Communication Complexity				
Coordination Complexity				
Autonomy				
Reliability				
Adaptability				



2.4.4 A Hybrid Control Architecture

In our application, we combine hierarchical control architecture and module control architecture together to form a hybrid control architecture. A few replications for critical agents are added into this control architecture to provide fault tolerance capability efficiently and flexibly.

2.4.4.1 Replication Ring

The idea of organizing a few replications of one agent in a flexible and robust way to improve the reliability of the whole system is originated from one characteristic of the Java Agent Development (JADE) framework. JADE is the chosen development environment for implementation of the proposed methodology in this dissertation and will be further discussed in the following context.

If an important control agent, such as the highest level agent in the multi-agent system, is damaged or unavailable to its lower level agents, the whole system will lose global control even though the subsystems can work according to its available information. In order to keep the whole system working as well as before when a failure of a significant control agent occurs, a few replications are created and arranged in a robust, efficient way to insure automatic reconfiguration to take place when necessary. Similar to the idea of fault-tolerance with replicated main containers in JADE, it starts with many replications of a significant control agent as needed. The replicated agents are software based and modularized, thus it is cheap to implement. All of the replications arrange themselves into a logical ring. Whenever one of the replications fails, the others will notice and act accordingly by using cross-notification. Agents connecting to the failed replication will be able to connect to some other replications and keep all of the information the same as before the damage happens.

Using the same example as shown in Figure 11, three replications for the critical agent are established. The resulting configuration is shown in Figure 14. In this configuration, we see four replications of a critical agent arranged in a ring. There are interactions between two neighbored nodes. For example, if critical agent replication 1 fails, its neighbor nodes: critical agent replication 2 and critical agent replication 3 will detect this and inform all other available replications such as replication 4 in this case. Then a new smaller ring consisting of the remaining critical agent replications are established.

In the ring configuration, all of the replications are identical and one of them is chosen randomly as an active replication to implement the functions of this critical agent. Other replications work as bridges but can be updated to be consistent with the active replication. Agents connecting to the failed critical agent replication (agent 2 in this case) can be arbitrarily spread among the available replications (agent 2 is reconnected to critical agent replication 4 in this case). This reconnection is completely automatic. The isolated agent that detects its critical agent replication unavailable will attach itself to another available replication. One way to achieve this is that each agent connecting to this critical agent needs a memory space storing all of the replications information in the system. This memory will be updated when a new configuration of the ring is rebuilt. Such an approach avoids generating notification traffic towards peripheral agents but assumes a fixed list address of critical agent replications. Another way is through function matching: initially, the peripheral agents do not store anything, but have some functions/characteristics as well as the critical agent replications do. During run time, each agent broadcasts its state and a role matching process for other agents is used to interact with their environments. This approach may cause communication traffic if a lot of agents are involved in one system. In summary, the updating process can be described as follows: once the new ring configuration is formed and the new active replication will

broadcast/send the new list of replications to every agent connecting to this critical agent. The rebuilt configuration after critical agent replication 1 fails is shown in Figure 15.

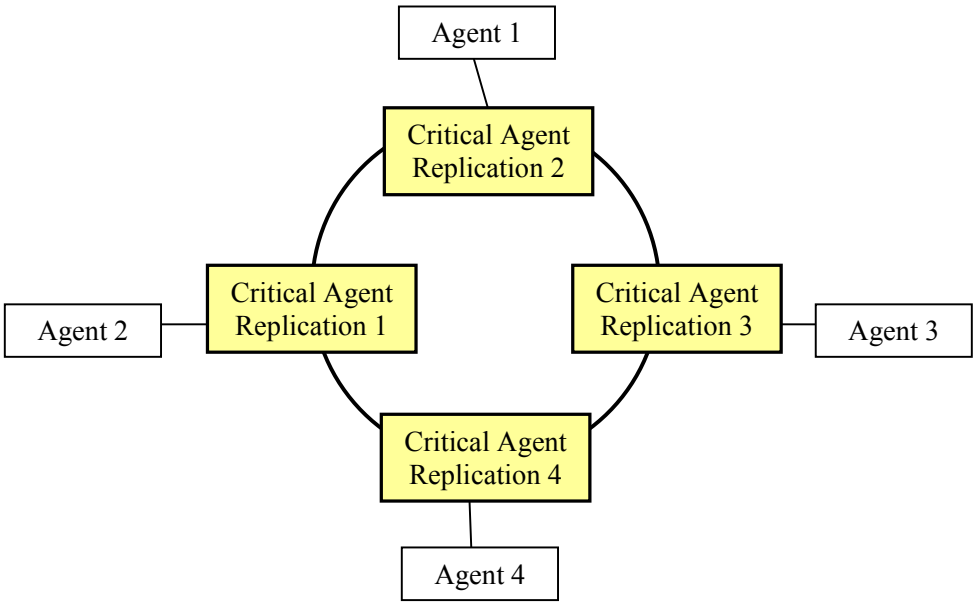


FIGURE 14 LOGIC RING CONFIGURATION FOR REPLICATIONS OF AN CRITICAL AGENT

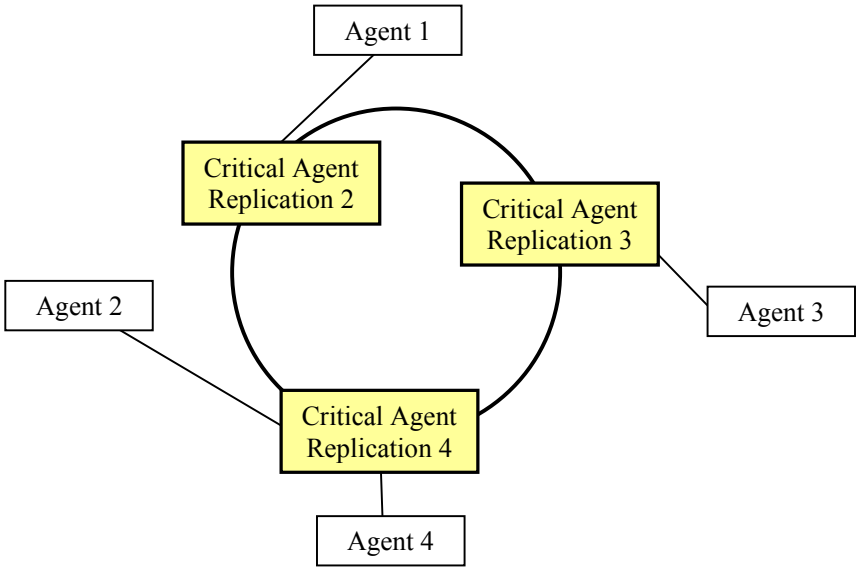


FIGURE 15 REBUILT RING CONFIGURATION OF REPLICATIONS AFTER REPLICATION 1 FAILS

2.4.5 Establish Hybrid Control Architecture Process

As mentioned before, hybrid control architecture, combining hierarchical control architecture and module control architecture together with critical agent replication rings, is proposed to establish a control system for large-scale complex systems. Herein, a summary of this architecture and the detailed description of the process are introduced. First the system is decomposed hierarchically, and then the components for each subsystem are chosen as a module. It simplifies the process of decomposition and design of the communication network. At the same time, it retains the advantages of modular architecture. It is important to keep in mind that the quantities of the layers for hierarchical control design depend on the structures of the physical model and the requirements of the control system.

Three fundamental steps: decomposition, abstraction and organization originally proposed by Booch [34] and Brooks [35] for object-oriented programming can be used to establish the framework of agent-based-control system here. In this paper, two additional steps, combined with these three steps, are used as a formal procedure to establish hybrid architecture of a multi-agent-based control system for large-scale complex systems. This hybrid architecture is clear, reliable, robust and easy to imbed a distributed inference engine to make it capable of handling incomplete and noisy information. What inference engine will be used and how to embed the chosen inference engine will be discussed in the following chapters of this dissertation.

2.4.5.1 Step 1: Decomposition

The purpose of this step is to divide the large complex system into smaller, more manageable pieces and address each piece in a relatively isolated manner. A system can be decomposed spatially by its components' spatial positions, or by using function analysis, or by using both of them simultaneously, depending on the structures of the physical system and the objectives of the control system. For example, it is reasonable to

divide a government's department into several bureaus by using function analysis. However, a power system of a building would be better divided according to the positions of its components. For a ship's chilled water system, considering its highly distributed components, both the spatial positions and functions of its components could be used to give a clear decomposed structure.

There are two processes to do control system decomposition: top-down approach and bottom-up approach [36].

- For the top-down approach, a centralized system model is first explicitly constructed and then it is decomposed into several subsystems by using structural properties presented in the system model. This method had been implemented in [36-38].
- For the bottom-up approach, there is no explicit centralized model. A virtual and conceptual model may be used as a reference. Subsystems are formed firstly and the designers check the relations between the subsystems to form higher level systems. This approach is adopted in [37-39].

These two processes can be combined in certain way. For example, first, establish a rough centralized model according to the information from subsystems; second, decompose the rough centralized model into clearer and more convenient subsystems; third, check the relations between the subsystems to refine higher level systems.

In this step, it is important for the control system designer to work closely with experts who are familiar with the physical system model. The use of pictures to show the decomposition and rough interactions among different pieces for the controlled system will give a clear view for the following two steps. Each piece from the decomposition can be addressed relatively independently to make the logic of each portion clearer, easier to understand and thus provides a result closer to the actual system.

2.4.5.2 Step 2: Abstraction

In this step, the aim is to design the internal logic and interface for each piece which can satisfy the autonomy requirement of individual agent. Munroe and Luck claimed that three sets of motivations: **Domain Motivations**, **Constraint Motivations**, and **Social Motivations** are sufficient enablers of autonomy in three key areas of agent operation [37].

- **Domain Motivations** represent the concerns and tasks that make up the agent's functional role in the system it belongs to.
- **Constraint Motivations** control the ways in which domain motivations are satisfied. It is similar to the concept of control constraints in a conventional way.
- **Social Motivations** determine the manner in which an agent interacts with other agents. The interface of each agent mainly depends on its relationships with other agents inhabiting in the same system.

To be more specific, first of all, the designers need to know the objective and structure of information that they can obtain. A relationship between its inputs and its outputs should be established and such a relationship has to satisfy its objective and constraints. Since each portion is relatively small, it is easy to implement complex control methods in a relatively short period of time. For example, the highest layer of the ship chilled water system needs information from the three second layer agents, i.e., the state of the whole chilled water resource: working normally, idle or damaged. It does not require any detail information on either of the two resources as long as one of the two resources is functioning properly. Similarly, the highest layer needs the state of each service load but not the state of each valve for a specific service load. More details about the ship's chilled water system will be provided in the application part of this dissertation. This portion of the process divides a big and complex task into smaller subtasks, and each

subtasks can be divided further. This characteristic will dramatically reduce the complexity of the design of the internal logic for each portion of the system. Herein, each section includes four parts: belief module, desire module, strategies planning module and communication module. The belief module acts as a diagnostic part to analyze the input data and infer the states of that agent as well as the state of its lower level agents. It then provides the information to its planning and commanding part which makes control strategies according to the information from the desire module. Through the communication module, it sends the control strategies to the controlled system.

2.4.5.3 Step 3: Organization

The objective of this step is to define and manage the interrelationships among various portions of the system. This step is the most challenging part in the procedure. However, if the interactions between the agents are sparse, the difficulty of establishing the relationships among various sections is alleviated considerably. Fortunately, since a complex system frequently takes the form of a hierarchy, a component in a subsystem could interact directly with components in another subsystem in a sparse way. Therefore, generally there are intensive interactions between the components in the same subsystem and sparse interactions between components in different subsystems. Furthermore, most of the communications between components in different subsystems can be conducted through a higher level agent instead of communicating with each other directly to make the establishment of relationships easier to organize. Each agent has its inputs and outputs from the second step (abstraction). The third step (organization) determines where the inputs come from and where the outputs go. By using an analogy to Object-Oriented Programming (OOP), each agent can be taken as a class. The inputs and outputs for each agent can be listed by using some conventional and understandable naming strategy, to help clarify the interrelationships among the agents. Some guidelines for OOP are similar

and can be adopted for the design of agent-based control systems, for example, the guidelines provided by Eckel for OOP [38].

Schillo and Fischer proposed a framework for definition of multi-agent organization: holonic multi-agent systems [39]. The basic idea of a hononic multi-agent system is that each part (holon) is a self-similar or fractal structure that is stable, coherent and consisting of several holons as sub-structure, and is itself a part of a greater whole. Holonic concept has three main advantages [39]. First, it preserves compatibility to multi-agent systems by addressing every holon as an agent. Second, it is a way of introducing recursion to the modeling of multi-agent systems for a large-scale complex system. Third, it does not force any social constraints for any individual agent. The idea of a holonic multi-agent system is similar to the ideas of hierarchical multi-agent structure and module multi-agent structure but more general. Schillo and Fischer also provide a set of conceptual level rules and operations for holons as well as the connection between different holons and agents' autonomy and objectives for multi-agent organization.

2.4.5.4 Step 4: Add Auxiliary Agents

This step is not necessary if the first three steps produce properly matched interfaces. There are three types of matching to consider when designing an agent-based control system:

- The interfaces between the different control agents must be matched.
- The interfaces between the control agents and the physical model must be matched.
- The interfaces between the control system and some other systems must be matched, e.g., the controllers for the ship's chilled water system must interface with the power system since they need to provide it with electric power to operate.

However, the control system and the physical system model are designed by different groups, therefore in most situations, their interfaces do not match exactly and some auxiliary agents need to be included to allow proper transformation of information between the entities. For example, one agent sends out an array of data in a specific order, but another agent will need some elements from the array or need the whole array in a different sequence. In this case, an input transformation agent may be needed for the second agent or an output transformation agent for the first agent. This will reduce the complexity of the two interdependent agents and allow the designers to focus on their internal logics. Another example of auxiliary agents needed is the preprocessing of data with uncertainty or the use of a separate agent to store some extreme situations with their corresponding planning to avoid disastrous consequences for ensuring the stability of the macro behavior of the system. Therefore, auxiliary agents can make the control system clearer and more robust. All of the agents that are not shown in the decomposition result can be included under the group of auxiliary agents.

2.4.5.5 Step 5: Create Replication Rings for Critical Agents

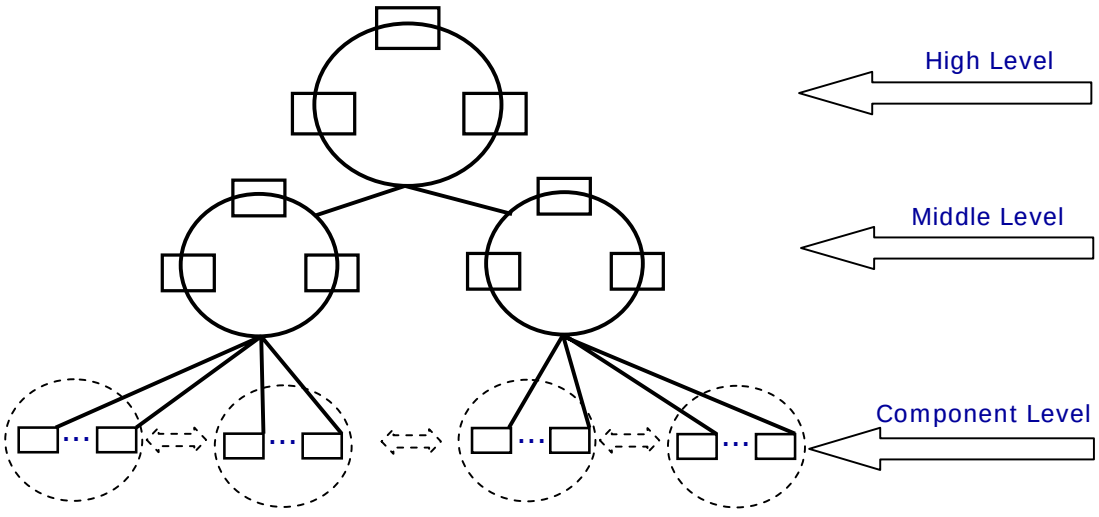


FIGURE 16 A HYBRID CONTROL ARCHITECTURE WITH REPLICATION RINGS

Choose critical agents according to a specific application and its corresponding decomposition schema and create as many replications as necessary for each critical agent. Add more functions to those replications such as monitoring its neighbor replications, communicating with other replications, forming new ring configurations among the replications, etc.

After the five steps proposed in the above context, notional hybrid control architecture with replication rings is shown in Figure 16.

2.4.6 Hypothesis 1.1 and Hypothesis 1.2

In summary, two hypotheses arise corresponding to research question 1.

- ***H1.1: A hybrid distributed multi-agent based control architecture is scalable, flexible and reusable.***

- ***H1.2: By using replication logical rings for critical agents, a hybrid distributed multi-agent based control architecture provides robustness of a control system to partial damage.***

2.5 Agent Development Platforms

Establishing a multi-agent based control system based on IP network from scratch takes a lot of time and requires expertise in network programming and familiarity with standardizations between different devices. Fortunately, there exist various types of software which make the design of distributed agent based control systems much more convenient than doing that from scratch. The control designer can focus on control methods, instead of message transport protocol design or standardizing the controllers.

Herein, a simple introduction is given for some agent application software platforms.

JACK™, is an environment for building, running and integrating commercial-grade multi-agent systems by using a component-based approach. JACK™ is based upon the company's research and development work on software agent technologies. The JACK™ Agent Language is a programming language that extends Java with agent-oriented concepts, such as agents, capabilities, events, plans, agent knowledge bases (databases), resource, concurrency management, etc.

3APL (An Abstract Agent Programming Language) is a programming language for implementing cognitive agents. It provides a programming structure for implementing agents' beliefs, goals, basic capabilities (such as belief updates, external actions, and communication actions) and a set of practical reasoning rules on which agents' goals can be updated or revised. The 3APL programs are executed on the 3APL platform. Each 3APL program is executed by means of an interpreter that deliberates on the cognitive attitudes of that agent. It has a rule-based reasoning engine to process deliberation.

JADE (Java Agent DEvelopment framework) is an open source software framework which is developed and distributed by Telecom Italia. JADE is probably the most wide-spread agent-oriented middleware in use today [40]. JADE is fully implemented in Java and complies with the FIPA (Federal Intelligent Physical Agent) specifications. JADE includes two products: a FIPA-compliant agent platform and a package to develop Java agents. JADE is made of numerous Java packages which provide application programmers with both ready-made pieces of functionality and abstract interfaces for custom application. The mission of FIPA is "*the promotion of technologies and interoperability specifications that facilitate the end to end interworking of intelligent agent systems in modern commercial and industrial settings*" [41]. The work of FIPA started in 1997 and its primary specifications became standardized in 2002. FIPA continues its work on modeling, methodology, semantics & services of physical agents design specifications. Currently, FIPA mainly focuses on agent lifecycle management,

message transport, message structure, inter-agent interaction protocols, ontologies, security, etc. According to the mission of FIPA, it is a set of specifications for promoting the technologies of agents, thus it does not limit the internal structure of agent design. Any agent development environment complying with FIPA will not conflict with any specific applications.

JADE simplifies the implementation of multi-agent systems through an agent-oriented middle-ware and a set of graphical tools that support the debugging and deployment phases. JADE provides a platform which is flexible and convenient for different users to plug in some add-ons according to different applications. JADE supports platform fault-tolerance, mobile agents and distributed platforms which can be located on different machines, or devices with different operating systems. Figure 17 shows the FIPA agent management reference model which JADE complies with.

The agent platform can be distributed across machines (which do not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can even be changed at run-time by moving agents from one machine to another when required. JADE is completely implemented in the Java language and the minimum system requirement is version 1.4 of JAVA (the run time environment or the JDK), which is a free download from Sun Corporation. JADE is more flexible for different applications. In JADE, the main container is the core node which provides the capabilities for managing all containers and agents (agent identities, agent descriptions, message transport protocols, etc), hosts the platform AMS agents and provides the platform Default DF agent, as shown in Figure 18. JADE supports a fault tolerant platform, which means it can use some backup main containers automatically and keep the whole agent system running as the main container has never been shut down if the platform of the main container has been damaged for some reasons.

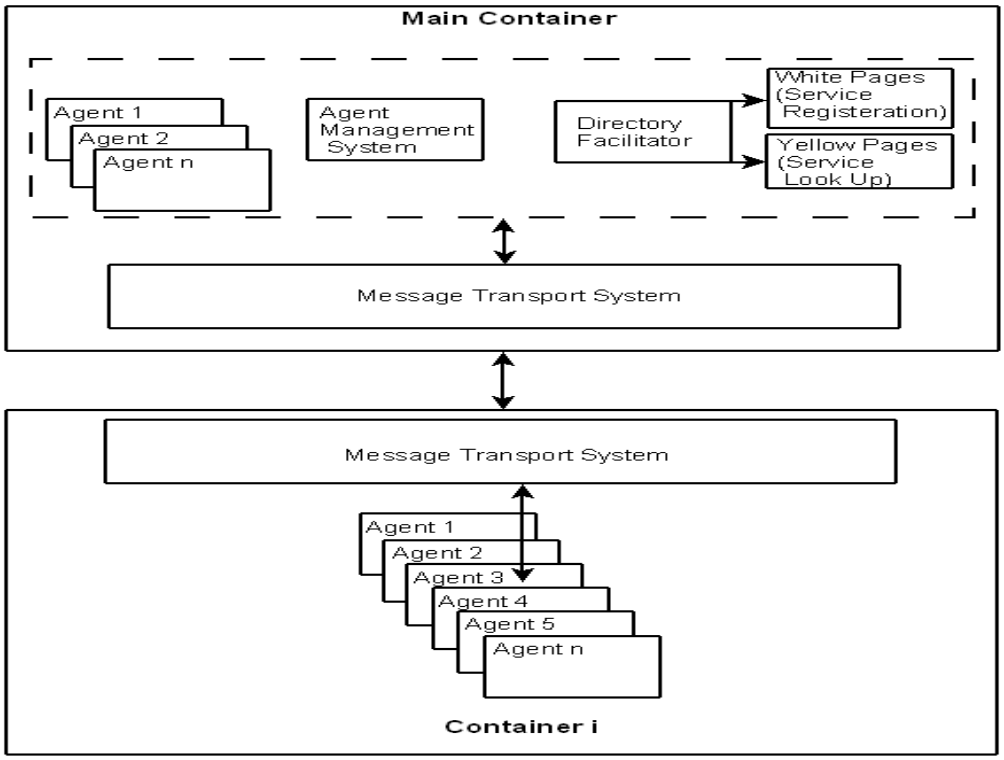


FIGURE 17 FIPA AGENT MANAGEMENT REFERENCE MODEL

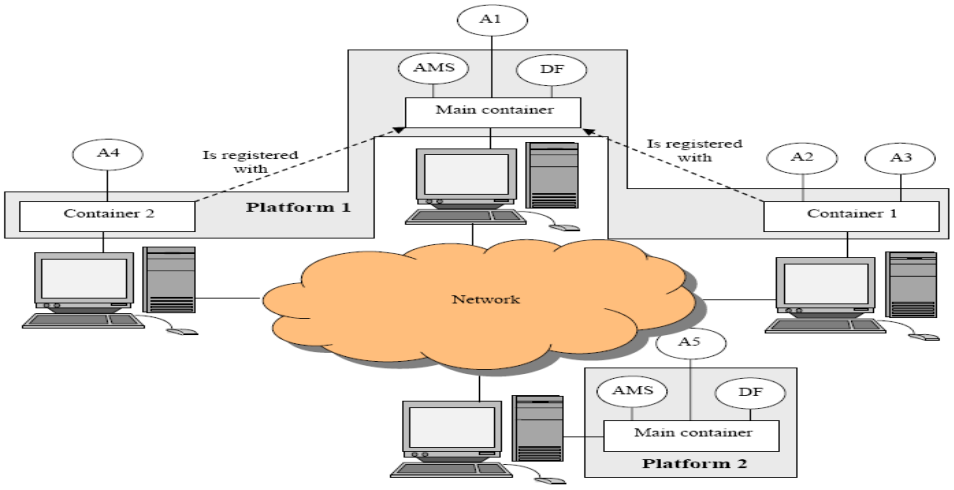


FIGURE 18 CONTAINERS AND PLATFORMS [42]

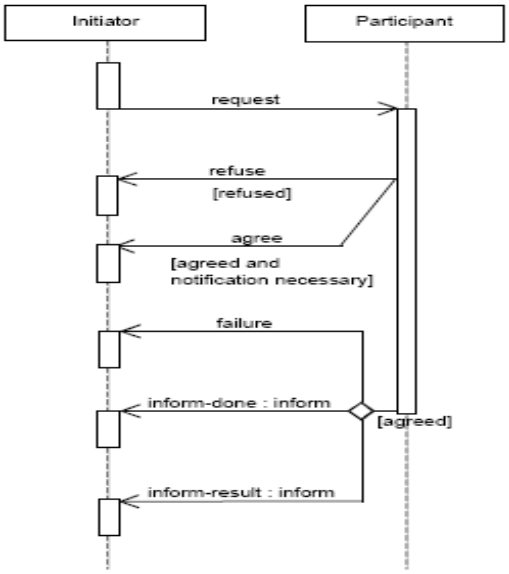


FIGURE 19 FIPA REQUEST INTERACTION PROTOCOL[43]

In JADE, an agent can be in one of five states after it is created: active, waiting, suspended, transited and killed. An agent can be initialized when it is created. Each agent has several behaviors. There are three types of behaviors: one time shot behavior, cyclic behavior and ticker behavior. All of the behaviors are stored in the behavior pool. The agent executes its behaviors from its behavior pool sequentially as long as its state is active. Each agent has its own thread, thus all of the agents independently execute their behaviors in parallel.

Jade also supports many standardized interaction protocols specified in FIPA, such as FIPA-Contract-Net, FIPA-Propose, FIPA-Subscribe, FIPA-Request, FIPA-Broker, etc. An interaction sketch of FIPA Request Interaction Protocol is shown in Figure 19. All of these interaction protocols can be used as standard templates to build agent intelligent conversations to reduce the programming time for the agent designers.

JADE is very comprehensive and flexible. The latest JADE version is JADE 3.6.1 released on 4th November 2008, and it continues to be enhanced. However, JADE is not

“drag and drop” software, and it needs a lot of code writing for specific agent internal structure design.

CHAPTER III

PRELIMINARY KNOWLEDGE OF BAYESIAN NETWORKS

In this chapter we give a simple introduction to the basic principles about probabilistic theories and Bayesian Networks. These basic principles will help readers understand the distributed probabilistic inference engines. Distributed probabilistic inference engines are important parts of this dissertation in multiple agent based control systems for large-scale complex systems with significant uncertainties. In the first part of this chapter, the preliminary knowledge of probabilistic theories is reviewed. In the second part of this chapter, an introduction to Bayesian networks is given. Finally, an efficient algorithm for belief updating in Bayesian networks is introduced in this chapter.

3.1 Preliminary Principles of Probability Theories

3.1.1 Basic Concepts

An Event's Probability is the value of one outcome over all possible outcomes in an experiment. A probability satisfies at least five conditions:

$0 \le P(V) \le 1$, where V is a event.

$P(\varnothing) = 0$, where $\varnothing$ is an empty set.

$P(V) = 1$, if and only if $V = S$, where S is the complete event set.

$P(\bigcup_{i=1}^{\infty} V_i) = \sum_{i=1}^{\infty} P(V_i)$ if and only if for $\forall i, j$, $V_i \cap V_j = \varnothing$ (V_i, V_j are mutually exclusive).

$P(V^c) = 1 - P(V)$, where V^c is the complement event set of V .

A Conditional Probability is an event probability based on knowing that another event occurred. For example, throwing a dice would get "1" with probability 1/6 if no any prior information is available. Now, if we know the number is less than 4, the thrown dice will be "1" with probability 1/3. The conditional probability of V_1 given V_2 is expressed as $P(V_1 | V_2)$.

Independence, event V_1 is independent of V_2 if $P(V_1 | V_2) = P(V_1)$.

Event V_1 and event V_2 are ***Conditional Independent*** for $V_3 = v_3$ if $P(V_1, V_2 | V_3 = v_3) = P(V_1 | V_3 = v_3) P(V_2 | V_3 = v_3)$. Conditional independence is a very important concept in Bayesian networks.

Joint Probability, for a give set of variables $\{V_1, \cdots, V_n\}$, the joint distribution of $\{V_1, \cdots, V_n\}$ is the distribution of the intersection of all of the events $\{V_1, \cdots, V_n\}$, i.e., the probability of events $V_1, \cdots, V_n$ occurring together.

For a set of *n* discrete random variables $\{V_1, \cdots, V_n\}$, the joint probability mass function is $P(V_1 = v_1, \cdots, V_n = v_n) = P(V_1 = v_1 | V_2 = v_2 \cdots, V_n = v_n) P(V_2 = v_2 \cdots, V_n = v_n)$, where $\sum_{v_1} \cdots \sum_{v_n} P(V_1 = v_1, \cdots, V_n = v_n) = 1$.

For a set of continuous random variables $\{V_1,\cdots,V_n\}$, the joint probability density function is $f(V_1=v_1,\cdots,V_n=v_n)=f(V_1=v_1\,|\,V_2=v_2\cdots,V_n=v_n)\,f(V_2=v_2\cdots,V_n=v_n)$, where $\int_{v_1}\cdots\int_{v_n} f(V_1=v_1,\cdots,V_n=v_n)dv_1\cdots dv_n=1$.

For independent events V_1 and V_2, there joint probability is $P(V_1,V_2)=P(V_1)P(V_2)$.

Marginal Probability Distribution

For a set of discrete random variables $\{V_1,\cdots,V_n\}$ with the joint probability mass function $P(V_1,\cdots,V_n)$, the marginal probability mass function of $V_1,\cdots,V_m$ is $P(V_1=v_1,\cdots,V_m=v_m)=\sum_{v_{m+1}}\cdots\sum_{v_n}P(V_1=v_1,\cdots,V_n=v_n)$.

For a set of continuous random variables $\{V_1,\cdots,V_n\}$, the joint probability density function is $f(V_1=v_1,\cdots,V_n=v_n)$, then the marginal probability density function of $V_1,\cdots,V_n$ is $f(V_1=v_1,\cdots,V_m=v_m)=\int_{v_{m+1}}\cdots\int_{v_n} f(V_1=v_1,\cdots,V_n=v_n)dv_{m+1}\cdots dv_n$.

Partition of a Sample Space S**,** a set of events $\{V_1,\cdots,V_n\}$ form a partition of a sample space if $V_1,\cdots,V_n$ are mutually exclusive and $\bigcup_{i=1}^{n}V_i=S$.

3.1.2 Bayesian Theorem and Potential

Bayesian Theorem: assume $\{V_j\}$ forms a partition of the whole event space, then for any V_i in the partition, $P(V_i\,|\,W)=\dfrac{P(W\,|\,V_i)P(V_i)}{P(W)}$, where, W is a happened event and $P(W)=\sum_i P(W\,|\,V_i)P(V_i)$.

A ***Potential*** is a real-valued table over a domain of finite variables.

Potentials are used to specify real-valued non-normalized probability tables. Potentials can specify prior probabilities, joint probabilities, conditional probabilities or any combinations of these. Every stochastic variable defined in a potential has an attached domain that is defined over the states of the stochastic variables. The domain for a potential ϕ is important and it is denoted as $dom(\phi)$. Potential with discrete finite domain is assumed through this dissertation and it has the following properties:

Assume $\phi_1, \phi_2,$ and ϕ_3 are three sets of potentials, then

Combination: $dom(\phi_1\phi_2) = dom(\phi_1)dom(\phi_2)$.

Commutative: $\phi_1\phi_2 = \phi_2\phi_1$.

Associative: $(\phi_1\phi_2)\phi_3 = \phi_1(\phi_2\phi_3)$.

Unit existence: the potential over an empty set is 1, so $pot(empty\ set)\phi = 1 \cdot \phi = \phi$.

A potential can be marginalized (it is similar to the probability marginalization as defined before). For example $\phi^{\downarrow W} = \sum_{dom(\phi)\setminus W} \phi$ is a new potential over W. This process is also called projection in some literatures.

Probability as a special type of potential has a few more properties:

Unit property: $\sum_V P(V) = 1$, $\sum_V P(V \mid W) = 1$ or $\sum_V P(V, W) = P(W)$.

The distributive law: if $W \cap dom(\phi_1) = \Phi$, then $\sum_W \phi_1\phi_2 = \phi_1 \sum_W \phi_2$.

The distributive law is very important for decreasing the size of joint probability tables. For example, assume $dom(\phi_1)$ has two variables V_1 and V_2, where each has 3 states; $dom(\phi_2)$ has 3 variables V_3, V_4 and V_5, where each has 4 states. Calculate $\sum_{V_3}\sum_{V_4}\phi_1\phi_2$. The biggest joint probability table size is $3\times3\times4\times4\times4$ without using the distributive law; while the biggest joint probability table size is $4\times4\times4$ if using the distributive law. The distributive law will improve the efficiency of inferences, which will be seen more clearly in the following context of this dissertation.

3.2 Bayesian Networks

3.2.1 Introduction

Bayesian networks are graphical models representing cause-effect relationships among different events. It displays the logic way of how human being thinks. A graphical model consists of nodes and links. The nodes represent events and the links between nodes indicate cause-effect relationships. Links have directions. A link from X to Y is different from a link from Y to X. If a link from X to Y, X is the cause (parent) and Y is the effect (child) and vise versa. Each node can be continuous or discrete with finite number of states. All of the nodes are connected with each other directly or indirectly, otherwise, the network can be divided into two independent graphs. Such a graphical model can be used to simulate and evaluate how changes in some variables could affect the remaining nodes of the system. Sometimes, it is difficult to distinguish from which variable is cause and which variable is effect. Under such situations, one can choose either of them subjectively without much effect on the validity of the model.

Bayesian networks can readily handle incomplete data sets. In many real systems, it is difficult to get a complete data set or it is too expensive to get a complete data set. For many other modeling techniques, they need a complete input data set. If one of the input

data is not observed, the models will produce an inaccurate prediction/reasoning results, because they do not encode the correlations among the input variables. Bayesian networks are based on the dependencies of all of the system variables and encode the cause-effect relationships in a clear, instinctive and simplified way [44].

Bayesian network techniques can be used in two general fields: regression modeling and statistical inference. Bayesian networks can handle continuous random variables and discrete random variables or a hybrid situation. In this dissertation, we focus on statistical inferences with discrete random variables.

As a formal definition in [45] , a Bayesian network consists of the following elements:

- A set of variables and a set of directed edges between variables.
- The variables together has a finite set of mutually exclusive states.
- The variables together with the directed edges form a Directed Acyclic Graph (DAG) (A directed graph is acyclic if there is no directed path $V_1 \to V_2 \cdots V_{n-1} \to V_n$, such that $V_1 = V_n$).
- To each variable V with parents $V_1, V_2 \cdots V_{m-1}, V_m$, there is a potential table $P(V \mid V_1, V_2 \cdots V_{m-1}, V_m)$ attached.

3.2.2 Basic Types of Connections in Bayesian Networks

In Bayesian Networks, there are three types of connections: serial connection, diverging connection and converging connection.

Serial Connection: the variables are connected sequentially in one direction. Figure 20 shows a serial connection.

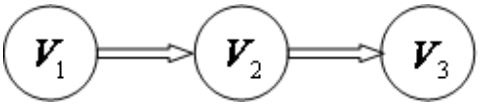


FIGURE 20 SERIAL CONNECTION

Diverging Connection: several variables have a common parent. Figure 21 shows a diverging connection.

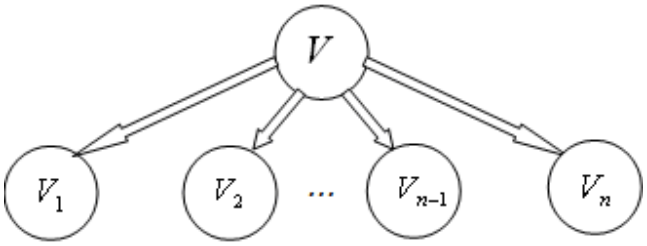


FIGURE 21 DIVERGING CONNECTION

Converging Connection: several variables have a common child. Figure 22 shows a converging connection.

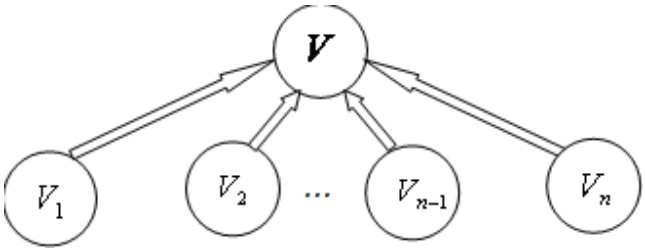


FIGURE 22 CONVERGING CONNECTION

The graph of Bayesian Network encodes dependencies between variables. Conditional independence can be determined by the graphical property of d-separation

D-separation: two sets of nodes U and V are d-separated in a directed graph by a third set W if the corresponding variable sets of U and V are independent given the corresponding

variable sets of W. Mathematically, a set U and a set V are d-separated by a set W if $P(U,V \mid W) = P(U \mid W)P(V \mid W)$.

D-separation is a very important concept and it is the base for efficient Bayesian network inference algorithms. D-separation can be obtained through manipulating the three types of connections in a Bayesian network.

- In the serial connection structure, as shown in Figure 20, if the state of V_2 is given (variable V_2 is instantiated), then the state of V_1 and V_2 are independent.

Mathematically: $P(V_1,V_3 \mid V_2 = v_2) = P(V_1 \mid V_2 = v_2)P(V_3 \mid V_2 = v_2)$.

- In the diverging connection structure, as shown in Figure 21, the influence between the children can be transmitted through the parent if the parent's state is unknown. However, if the parent is instantiated, the state of one child has no effect on the state of another child anymore.

Mathematically: $P(V_i,V_j \mid V = v) = P(V_i \mid V = v)P(V_j \mid V = v)$.

- For the converging connection structure as shown in Figure 22, it is a little bit more complicated to show how the d-separation is established between the parents. If nothing about the common child V is known, the parents of V are independent.

Mathematically: $P(V_i,V_j \mid V \textit{ is unknown}) = P(V_i \mid V \textit{ is unknown})P(V_j \mid V \textit{ is unknown})$.

3.2.3 Rules in Bayesian Networks

Before introducing the most important rule (Chain Rule) in Bayesian networks, we need to introduce evidences/observations and product rule first.

Evidence/observations are defined as a collection of findings. There are two types of evidences: hard evidence and soft evidence. Hard evidence on variable V is a

specification of the value of *V*, and soft evidence on variable *V* is a distribution on the values of *V*. Normally, in most of applications, dealing with hard evidences is enough and most of software can only handle hard evidences.

Product Rule: $P(AB) = P(A \mid B)P(B)$

Chain rule is formed by successively applying product rule. It is described in the following:

Chain Rule: for a Bayesian network, its overall space consists of $U = \{V_1, V_2 \cdots V_{n-1}, V_n\}$, then

$$P(U) = P(V_1, V_2 \cdots V_{n-1}, V_n) = \prod_{i=1}^{n} P(V_i \mid \textit{parents of } V_i) \tag{3.1}$$

Let $\underline{o}_1, \underline{o}_2, \cdots, \underline{o}_{n-1}, \underline{o}_n$ be sets of evidences/observations, then the joint probability including the observations is

$$P(V_1, V_2 \cdots V_{n-1}, V_n, \underline{o}_1, \underline{o}_2, \cdots, \underline{o}_{m-1}, \underline{o}_m) = \prod_{i=1}^{n} P(V_i \mid \textit{parents of } V_i) \prod_{i=1}^{m} \underline{o}_i \tag{3.2}$$

, and

$$P(V_1, V_2 \cdots V_{n-1}, V_n \mid \underline{o}_1, \underline{o}_2, \cdots, \underline{o}_{m-1}, \underline{o}_m) = \frac{P(V_1, V_2 \cdots V_{n-1}, V_n, \underline{o}_1, \underline{o}_2, \cdots, \underline{o}_{m-1}, \underline{o}_m)}{\sum_{V_1} \cdots \sum_{V_n} P(V_1, V_2 \cdots V_{n-1}, V_n, \underline{o}_1, \underline{o}_2, \cdots, \underline{o}_{m-1}, \underline{o}_m)} \tag{3.3}$$

The proof is shown in [45] page 22.

3.2.4 Probabilistic Inference Algorithms in Bayesian Networks

Although Bayesian networks simplify probabilistic inferences by using conditional dependencies, an exact inference in an arbitrary Bayesian network for discrete variables

is NP-hard [46]. There are some techniques for approximate inferences. For example, stochastic simulation uses the causal model to simulate the flow of impact. By running simulations which match the available observations for many times, an approximate inference based on the simulated runs are attained. However, an approximate inference through simulation is still NP-hard [47], especially when there are many undirected cycles in a network. The efficiency of belief updating in Bayesian networks is very important for probabilistic inferences. Establishing an efficient belief updating algorithm is fundamental to the application of Bayesian networks. Otherwise, it will be too slow to be used in practice. Here, a few methods are listed: Enumerative Algorithm, Global Joint Distribution Algorithm, Bucket Elimination Algorithm, Pearl's Algorithm, Lauritzen-Spiegelhalter (L-S) Algorithm, Hugin Algorithm and Shenoy-Shafer (S-S) Algorithm. In this dissertation, Hugin Algorithm will be used in the application. Hugin Algorithm is among the most efficient methods known for belief updating in Bayesian networks in state of the art. Here, a simple introduction is given and more detailed information on this algorithm is described in [45]. Another reason of introducing this algorithm is because it has strong relationships with Multiple Sectioned Bayesian Networks (MSBNs) (more detailed discussion for MSBNs will be presented in the next chapter).

3.2.4.1 A Notional Example

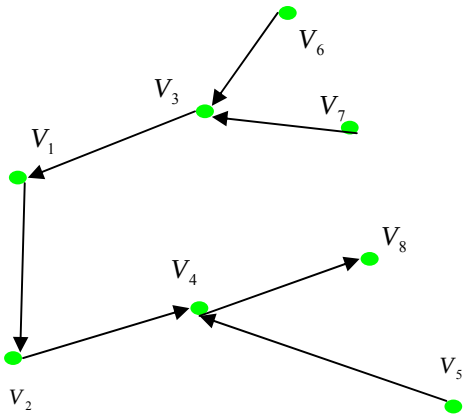


FIGURE 23 A NOTIONAL BAYESIAN NETWORK

In order to illustrate this method more clearly and conveniently, a notional Bayesian network model shown in Figure 23 is used for this whole introduction. There are 7 nodes in this notional example Bayesian network. The potentials specified for this network are: $\phi_1 = P(V_1 \mid V_3)$, $\phi_2 = P(V_2 \mid V_1)$, $\phi_3 = P(V_3 \mid V_6, V_7)$, $\phi_4 = P(V_4 \mid V_2, V_5)$, $\phi_5 = P(V_5)$, $\phi_6 = P(V_6)$, $\phi_7 = P(V_7)$, $\phi_8 = P(V_8 \mid V_4)$.

3.2.4.2 Hugin Belief Updating Algorithm

Hugin belief updating algorithm includes a few steps: moralization, triangulation, joint tree formulation, junction tree formulation and full propagation. Detailed descriptions of these steps are showing in the following:

Moralization

A moral graph of a Bayesian network is an undirected graph which connects any pairs of variables being members in any $dom(\phi_i)$ existing in the Bayesian network. For the notional example Bayesian network, the moral graph is shown in Figure 24. Compared to the initial Bayesian network, we can see that a link between V_6 and V_7 and a link between V_2 and V_5 are added because V_6 and V_7 have a common child as well as V_2 and V_5 .

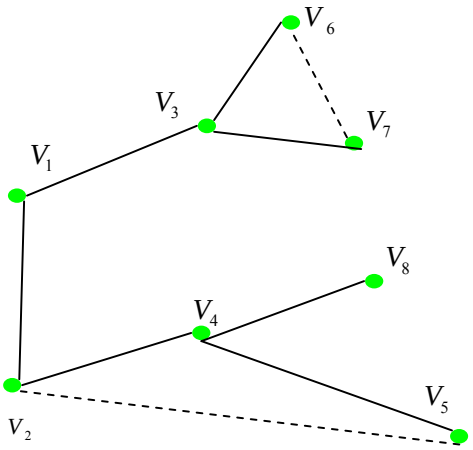


FIGURE 24 THE MORAL GRAPH OF THE NOTIONAL EXAMPLE

Triangulation

In order to introduce the idea of triangulation, first, the concept of perfect elimination sequence is introduced. When eliminating a node V_i in a Bayesian network, we work with the product of all potentials with V_i in the domain. The domain of this product consists of V_i and all of its neighbors in the moral graph. When V_i is eliminated, the resulting potentials has all ofV_i's neighbors in its domain and all of the variables in this new domain need to be connected pare wise. For example, when V_1 is eliminated, the new moral graph is shown in Figure 25. From this figure, we can see that a new link between V_3 andV_2 are introduced. The added link is called a fill-in. The introduction of fill-ins indicates that a potential of a new domain is presented when a variable is eliminated. Eliminating all of the variables in a network one by one forms an elimination sequence. Apparently, an eliminated sequence without introducing fill-ins (perfect eliminated sequence) requires less space than an elimination sequence that introduces fill-ins.

To continue, a few more concepts are introduced.

Complete Nodes Set: a set of nodes is complete if all nodes in this set are pair wise linked.

Clique: a complete set is a clique if it is not a subset of another complete set, i.e., the maximal complete set contains a set of specific nodes.

A Bayesian network may have multiple perfect elimination sequences. However, all of the perfect elimination sequences produce the same domain set, i.e., all of the elimination sequences have the same set of cliques. . For example, in the notional example, the eliminated sequence $V_8 \to V_4 \to V_5 \to V_2 \to V_1 \to V_3 \to V_6 \to V_7$ is a perfect elimination sequence with domain sets: $\{V_8, V_4\}$, $\{V_4, V_2, V_5\}$, $\{V_2, V_1\}$, $\{V_1, V_3\}$,

$\{V_3, V_6, V_7\}$, while the eliminated sequence $V_7 \to V_6 \to V_3 \to V_1 \to V_2 \to V_5 \to V_8 \to V_4$ is a perfect elimination sequence with the same domain set $\{V_3, V_6, V_7\}$, $\{V_1, V_3\}$, $\{V_2, V_1\}$, $\{V_4, V_2, V_5\}$, $\{V_8, V_4\}$ as well.

The proof is shown in [45]

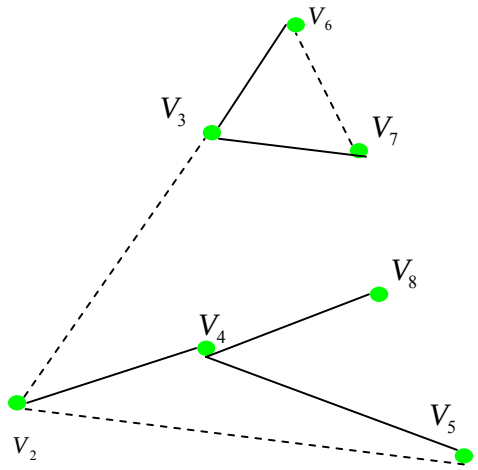


FIGURE 25 THE MORAL GRAPH AFTER NODE V_1 IS ELIMINATED

Triangulated Graph: an undirected graph with a perfect elimination sequence is called a triangulated graph.

Usually, it is NP-hard to determine the set of cliques in a graph. However, a heuristic procedure can be used to get the set of cliques for a triangulated graph. The procedure is shown in the following [45]:

- Eliminate a simplicial node V_i (nodes with a complete neighbor set are called simplicial, then this node with its neighbors denoted as F_{V_i} is a clique candidate.
- If F_{V_i} does not include all remaining nodes, go to step 1.

- Keep the clique candidates which are not subsets of any other clique candidates.
- The resulting set is the set of cliques.

3.2.4.3 Join Tree

Running Intersection Property [48]: let T be a cluster tree over a domain U. We say T has the running intersection property if whenever there is a variable $V_i \in (C_j \cap C_k)$ and V_i is contained in every cluster in the unique path in T between C_j and C_k, where C_j and C_k are two clusters in T.

Join Tree: Let G be the set of cliques from an undirected graph, and let the cliques of G be organized into a tree T. If T satisfies the running section property, then the tree T is a join tree.

An undirected graph is triangulated if and only if the cliques of this graph can be organized into a join tree.

The proof is shown in [45].

How to get a junction tree for a triangulated graph? Finn gives a detailed procedure about constructing a join tree for a triangulated graph as follows [45]:

- Establish a perfect elimination sequence by starting with a simplicial node V_i, then F_{V_i} is a clique.
- Continue eliminating all of the nodes which have only neighbors in F_{V_i} from F_{V_i}.
- Give F_{V_i} an index i according to the number of nodes being eliminated up to this point and denote of the remaining nodes in F_{V_i} as S_i. S_i is a separator for F_{V_i}.

- Choose a new simplicial node in the undirected graph with all of the eliminated nodes removed, and repeat step 2 until all of the nodes have been eliminated.
- Connect all of the cliques with their corresponding separators.
- Connect each separator S_i to a clique C_j with higher index j than i, such that $S_i \subseteq C_j$.

Then, the formed structure is a join tree for this undirected graph.

The proof is shown in [45].

Here, the notional example is used to illustrate this procedure in a more understandable way.

First, establish a perfect elimination sequence $V_8 \to V_4 \to V_5 \to V_2 \to V_1 \to V_3 \to V_6 \to V_7$ with the clique sets as $C_1 = \{V_8, V_4\}$, $C_3 = \{V_4, V_2, V_5\}$, $C_4 = \{V_2, V_1\}$, $C_5 = \{V_1, V_3\}$, $C_8 = \{V_3, V_6, V_7\}$, and their corresponding separators as $S_1 = \{V_4\}$, $S_3 = \{V_2\}$, $S_4 = \{V_1\}$, $C_5 = \{V_3\}$. The result is shown in Figure 26.

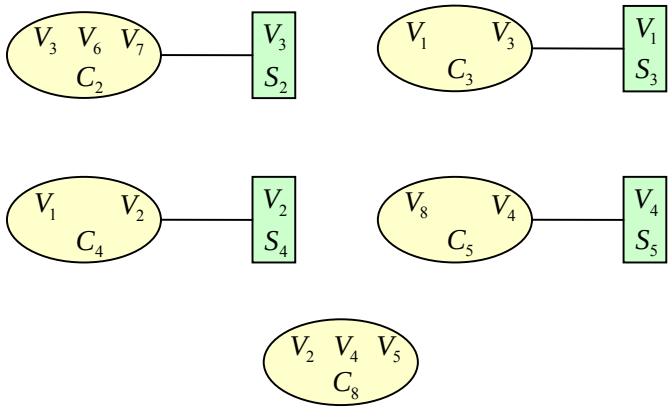


FIGURE 26 THE CLIQUES AND SEPARATORS AND INDICES FROM STEP 3 FOR THE NOTIONAL EXAMPLE

By implementing step 4, we get the connected rough graph joint tree for the notional example. The resulting graph is shown in Figure 27.

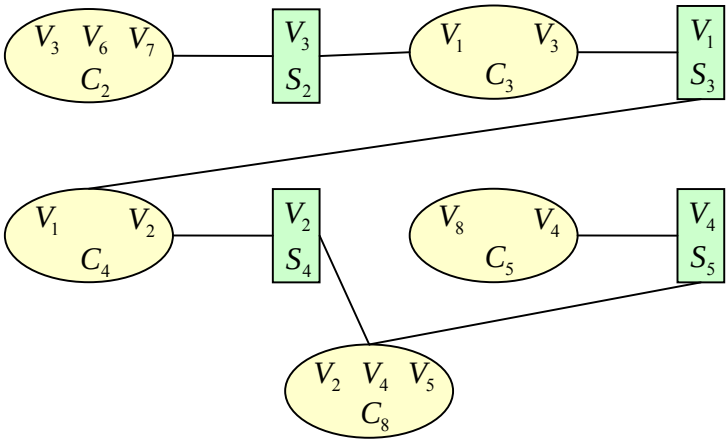


FIGURE 27 ROUGH JOINT TREE STRUCTURE FOR THE NOTIONAL EXAMPLE

Refine the rough joint tree by omitting the separators and connecting the cliques directly, we get a clear view of the joint tree for the notional example as shown in Figure 28. Apparently, the joint tree satisfies the running property.

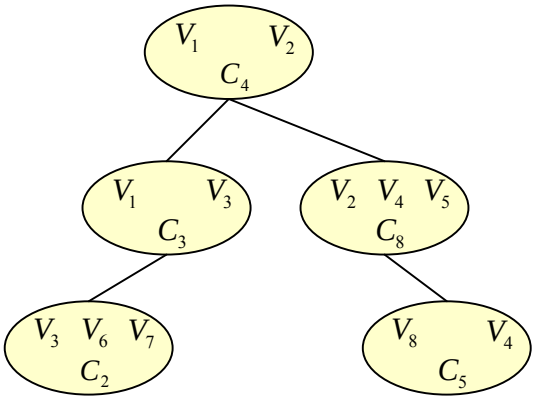


FIGURE 28 CLEAR VIEW OF THE JOINT TREE FOR THE NOTIONAL MODEL

Now, we will show how the information is propagated by using a junction tree, which highly improves the efficiency of belief updating.

3.2.4.4 Junction Tree

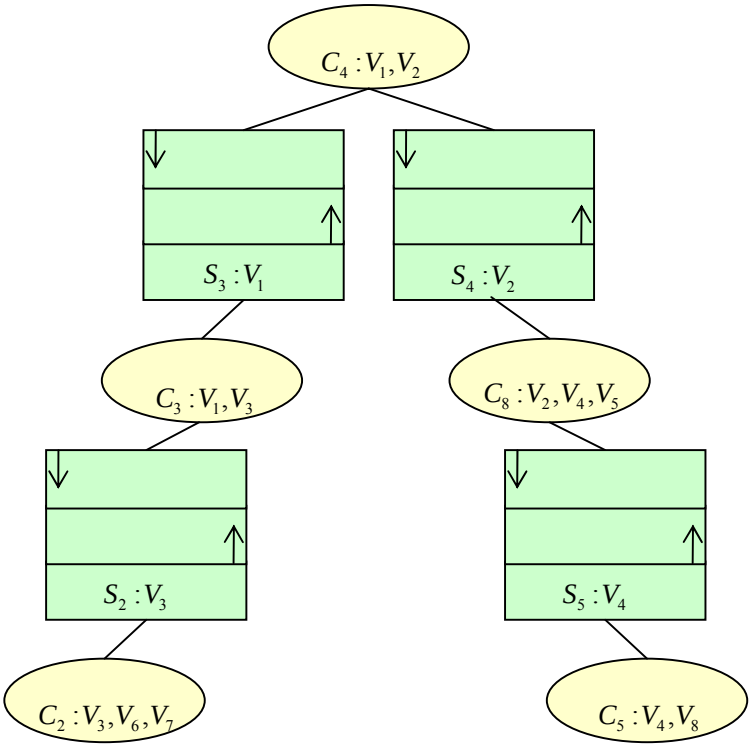


FIGURE 29 A JUNCTION TREE FOR THE NOTIONAL EXAMPLE BAYESIAN NETWORK

Definition[45]: Let G be a triangulated Bayesian network with a set of potentials Φ. A junction tree for G is a join tree for G with the following further structure: each potential $\phi \in \Phi$ is attached to a clique who contains $dom(\phi)$; each link has the appropriate separator attached; each separator contains two mailboxes: one for each direction.

The junction tree for the notional example is shown in Figure 29.

3.2.4.5 Belief Propagation in Junction Trees

For the notional example, assume we want to know the probability of $P(V_7)$. The first thing to do is to find a clique containing V_7. Such a clique is C_2 as shown in Figure 29. Take C_2 as a temporary root and collect the messages in the direction from the leaf cliques to the root cliques and store the messages in their corresponding message boxes.

For example, the message $\psi_5 = \phi_5^{\downarrow S_5}$ from clique C_5 is placed in the lower mail box of S_5; C_8 receives message ψ_5 and send C_4 a message $\psi_4 = (\phi_4\phi_6\phi_5)^{\downarrow S_4}$ which is based on its own potentials and its received messages. Similarly, C_4 sends a message $\psi^3 = (\phi_3\psi_4)^{\downarrow S_3}$ to C_3; C_3 sends a message $\psi^2 = (\phi_2\psi^3)^{\downarrow S_2}$ to C_2 which is the chosen root clique, and this process is done. Generalize this process as follows:

- Choose a clique containing the domain of the potentials which you want to get as a root clique.
- From the chosen root clique, trace the leaf cliques.
- The leaf cliques send messages to the chosen root clique and store all of the intermediate messages in the corresponding mailboxes on the way to the root clique.

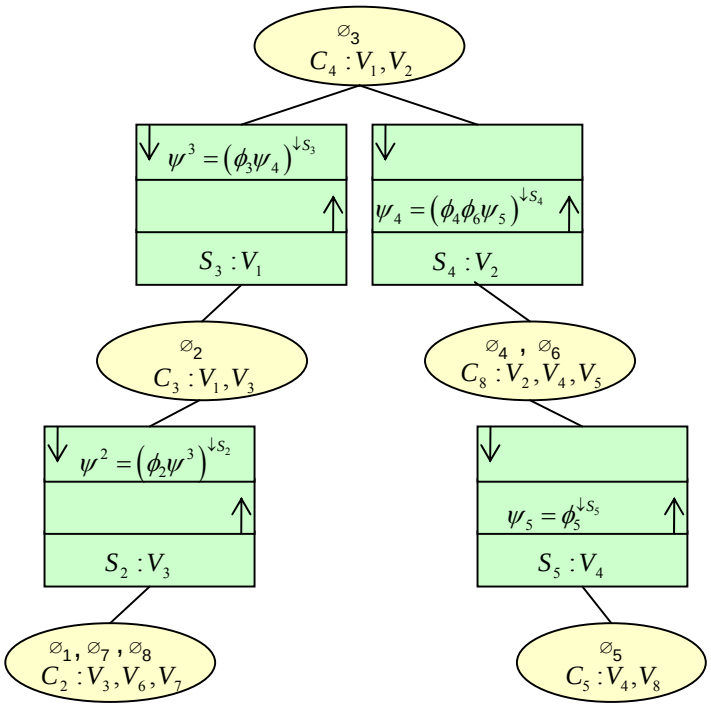


FIGURE 30 THE JUNCTION TREE OF THE NOTIONAL EXAMPLE AFTER THE COLLETING EVIDENCES PROCESS

Such a process is called ***Collect Evidence*** to the chosen root clique. After this process, any potential with the domain belonging to the root clique can be calculated based on the root clique's potentials and the root clique's receiving messages. For example, the probability of V_7 is $P(V_7) = \left(\phi_1\phi_7\phi_8\psi^2\right)^{\downarrow V_7}$.

However, by the collecting evidence process, we can only calculate the potentials with domain belonging to the chosen root clique. If we want to get potentials with other domains, we need to repeat this process again. Is there a way that we can calculate all of the potentials involving any domains in a Bayesian network without repeating the process again and again? We can prepare the junction tree for the calculations of all marginals' by sending messages in the direction away from a chosen root clique (This process is called ***Distributing Evidence***) after the process of collecting evidences to the chosen root clique [45]. The distributing evidence process is performed after the collecting evidence process, so for a clique sending messages away from the root clique, it should combine its own potentials and all of the messages it receives before its current action. Those messages include all of the messages received in the process of collecting evidence and the messages received in the process of distributing evidence before the current time. For example, from the chosen root clique C_2, it sends a message $\psi_2 = \left(\phi_1\phi_7\phi_8\right)^{\downarrow S_2}$ to C_3. Here the message is simplified from $\psi_2 = \left(\phi_1\phi_7\phi_8\psi^2\right)^{\downarrow S_2}$. We need to justify that this simplification does not lose any information. Here, we give a heuristic explanation: C_2 has gotten the message ψ^2 from C_3, which means C_3 has already had the information of ψ^2, so C_2 does not need to send C_3 such information which originates from C_3. For some other situation, suppose C_2 has another neighbor C_x which has send a message ψ^x to C_2, then C_2 need to combine ψ^x when sending a message to C_3. This message should be

expressed as: $\psi_2 = \left(\phi_1\phi_7\phi_8\psi^x\right)^{\downarrow S_2}$. Similarly, C_3 sends the message $\psi_3 = \left(\phi_2\psi_2\right)^{\downarrow S_3}$ to C_4; C_4 sends the message $\psi^4 = \left(\phi_3\psi_3\right)^{\downarrow S_4}$ to C_5; C_5 sends the message $\psi^5 = \left(\phi_4\phi_6\psi^4\right)^{\downarrow S_5}$ to C_8. The combination of the collecting evidence process and the distributing evidence process is called a ***Full Propagation***. The junction tree of the notional example after a full propagation described in the above is shown in Figure 31. Now the junction tree is ready to calculate any marginal involving any domains contained in any one clique.

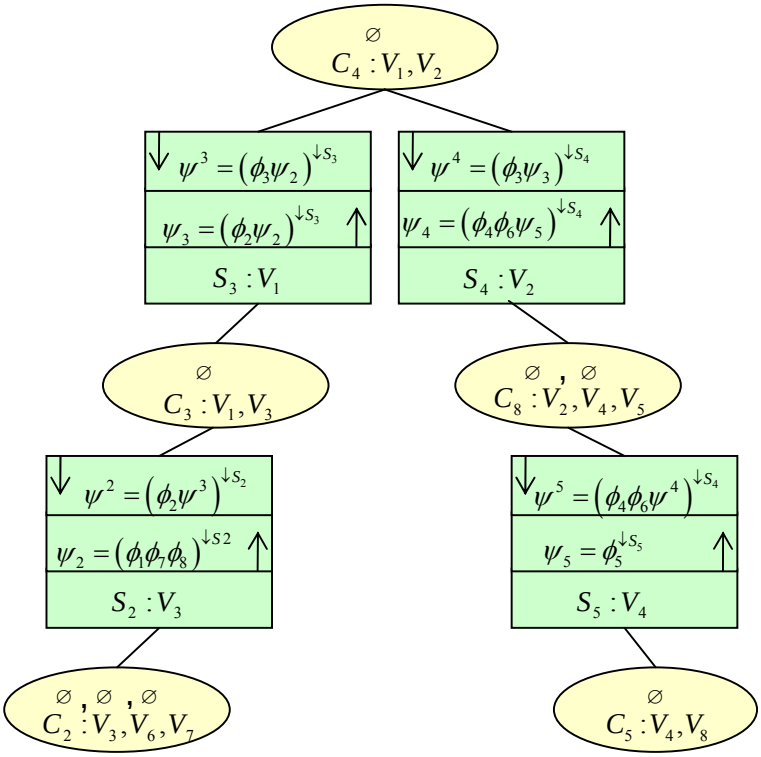


FIGURE 31 THE JUNCTION TREE OF THE NOTIONAL EXAMPLE AFTER A FULL PROPAGATION

A general process to calculate a potential with domain X is as follows:

- Find a clique C_i containing X.

- Find all of the messages that the clique C_i has received.
- Multiply all of C_i's potentials and its received messages.
- Project the multiplication down to X.

For example, we want to calculate the probability of V_7. First, we find clique C_4 contains V_2. Second, we find C_4 has received two messages: ψ_3 and ψ_4. Third, multiply the potential ϕ_3 of C_4 and the two messages to get $\phi_3\psi_3\psi_4$. Forth, project the multiplication down to V_2 and get the probability of V_2 as $P(V_2) = (\phi_3\psi_3\psi_4)^{V_2}$. Furthermore, since V_2 is also contained in one of the separators S_4, there is a slightly easier way to calculating $P(V_2)$ by multiplying the two messages in S_4: ψ_4, ψ^4 and projecting this multiplication down to V_2. We can prove that these two methods give the same result [45].

CHAPTER IV

DISTRIBUTED INFERENCE ENGINE

4.1 Introduction

The problem of keeping track of the state of a system over a certain time is generally called monitoring or filtering. For a dynamic system with uncertainty, the monitoring or filtering process is stochastic and the goal of inference is to maintain a probability distribution over the state of the system at each time point, based on the evidences/observations available up to that point[49].

There are a few reasons for a large-scale complex system to use a distributed dynamic inference engine for state estimations:

- It is impossible/difficult to get a complete data set.
- Knowledge of the system internal dynamics and its environment dynamics is insufficient.
- Available information is contaminated by numerical/system noises.
- The system is evolving over time.
- Information available is distributed.
- Capabilities of computation and communication for local processors are limited.

Using a big table listing all of the variables to do observation matching is not practical. For example, for a simple system with 10 variables, if each variable has 4 states, we need a table as large as 4^{10} to store all of the information. Normally, in a complex system, one

variable may just relate to a few limited variables. Once the relevant variable states are known, the state of the chosen variable will not be affected by other variables in the system. This idea is called d-separation which was defined in Chapter II.

In decomposing a large-scale complex system into a set of decoupled subsystems, a lot of information from experts, or knowledge from previous precious experiments indicating part interactions of the system will be lost. Using an inference mode to catch all of the available information and previous experience to determine current system states to help control the system is very important to maximize the efficiency of a control system.

How to use expert knowledge or previous experiment data? Generally, we call such a process as establishing an inference engine. In computer science, specifically the branches of knowledge engineering and artificial intelligence, an inference engine is a computer program that tries to derive answers from a knowledge base. It is the "brain" that expert systems use to reason about the information in the knowledge base for the ultimate purpose of formulating new conclusions. Inference engines are considered to be a special case of reasoning engines, which can use more general methods of reasoning [46-48, 50].

Now we know what an inference engine is. A research question corresponding to the second research aspect of this dissertation arises:

- ***Q2: Is there an inference engine that can handle uncertainties of large-scale complex systems?***

To make this research question clearer and more understandable, it can be decomposed into five smaller sub research questions:

- ***Q2.1: How does the inference engine handle incomplete and uncertain data sets?***

- ***Q2.2: How does the inference engine work for a distributed system?***
- ***Q2.3: How does the inference engine reach global consistencies if distributed?***
- ***Q2.4: How does the inference engine work for a dynamic system (structure dynamics and component state dynamics)?***
- ***Q2.5: How should the inference engine be integrated into the control architecture?***

Basically, a distributed inference engine includes four aspects:

- local information processing,
- partial intermediate information exchange,
- inference global consistency,
- self-organization due to partial damage.

A distributed inference engine can be modeled as a multi-agent system. Each agent represents certain local knowledge of subsystems. A reasonable and working distributed inference engine should have the following basic capabilities:

- If an agent is isolated from other agents, it can infer its local node states based on its local available information correctly (Local Intelligence and Independence).
- If some links between two agents are damaged, an agent can infer its local nodes based on its local measurements and the available messages it receives from its neighbors (Local Robustness and Optimization).
- If all of the connections among the agents are not damaged, through communication, the state estimations for all of the agents are consistent (Coordination and System Consistency).

- If some agents are missing or some links between two agents are damaged, the remaining agents can reorganize themselves into certain structures based on existing link quality. Coordinated inference can be initiated through the reorganized structures (Self-organization and Inference Automation).

In this chapter, several inference reasoning methods: cased-based reasoning, rule-based reasoning, model-based reasoning and Bayesian network reasoning are introduced and evaluated. After careful comparison of these reasoning methods, Bayesian network reasoning is chosen to be further investigated as a distributed probabilistic inference engine. There are three types of distributed Bayesian networks: Distributed Perception Networks (DPNs), Prior/likelihood Decomposable Models (PDM) and Multiple Sectioned Bayesian Networks (MSBNs). By further comparing these three distributed Bayesian networks, the MSBNs method with the most promising characteristics is chosen as a distributed probabilistic inference engine to be embedded into the multi-agent based control architecture established in Chapter II. Furthermore, automatic and distributed algorithms of formulation of MSBNs structures are also discussed in detail.

4.2 Case-Based Reasoning

Reasoning by finding similar cases in a predetermined past case base according to current inputs/observations is called Case-Based Reasoning (CBR) as shown in Figure 32. Each case typically contains one scenario of a system. CBR methods have in common the following process [51]:

- Retrieve the most similar case/cases, which match the current system observations the most. In this step, if several cases are chosen, logics/rules of combining them are required.

- Use the resulting case to try to solve the current problem. If some conflicts appear, revise and adapt the resulting case to a new case according to current system observations.
- Add this new case into the case base as a way of self extension and learning.

CBR works perfectly if the current situation matches one of the stored cases exactly. However, enumerating and storing all possible cases for a complex system is not practical. And if the case base is too big, the retrieving process for a similar case would be very slow. Since a case describes one scenario of the whole system, it would not be easy to implement in a distributed way.

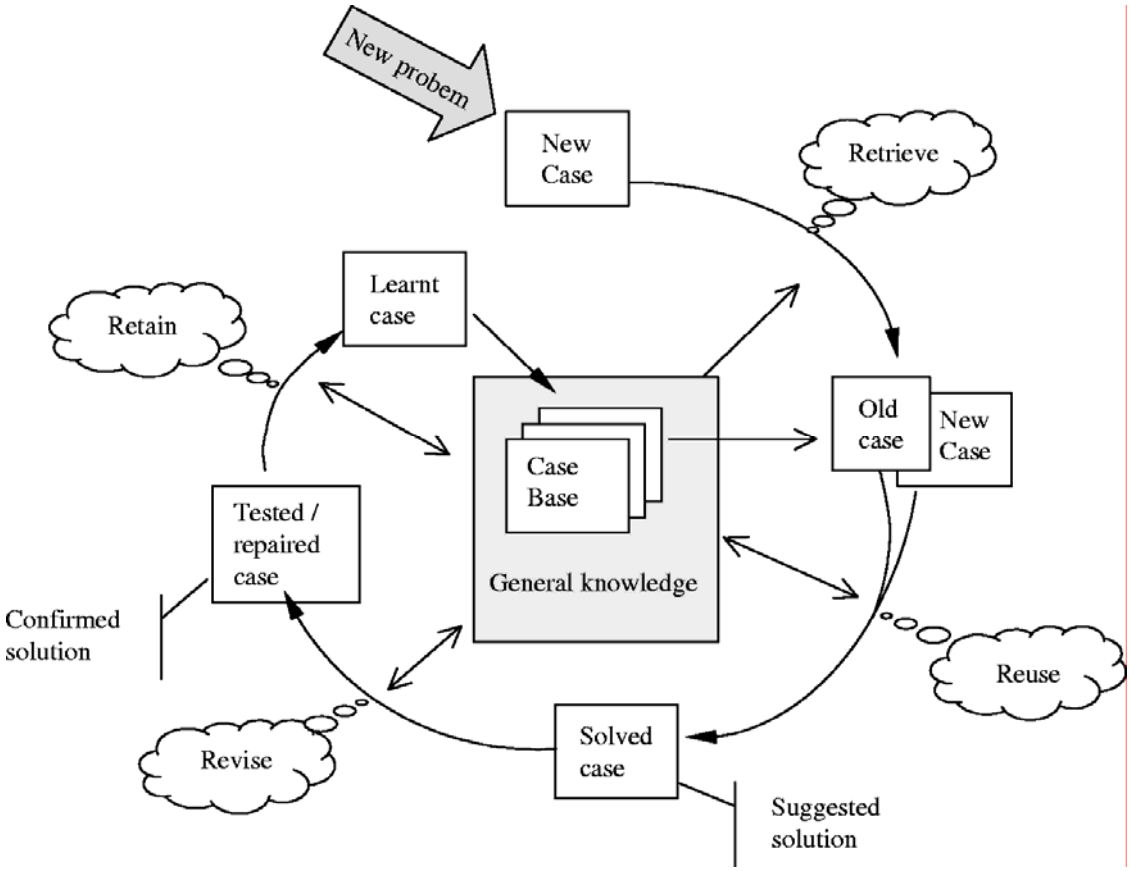


FIGURE 32 CASE-BASED REASONING [52]

4.3 Rule-Based Inference Engine

A rule-based engine is also called expert inference engine/shallow reasoning and is most often used in reasoning methodologies which are associative in nature. An expert system often consists of three parts: a knowledge base, an inference engine (searching strategies) and an user interface [53].

The knowledge base includes many facts and rules representing the knowledge about a particular system from the domain of expertise. A rule indicates a relationship between two facts. Its simplest format is:

$$IF\left(Conditions\right)\ Then\ \left(Facts\ or\ Actions\right) \tag{4.1}$$

Two algorithms of inference in rule based reasoning are often used: forward chaining and backward chaining [53]. Forward chaining starts with the available data and uses inference rules to extract more data [9, 54]. In other words, forward chaining is a top-down searching procedure. It searches the rule base and when it finds the rule IF part conditions are satisfied, it uses the THEN part as the conclusion. Backward chaining is the reverse. It is a bottom-up searching procedure. An inference engine using backward chaining would search the inference rules until it finds one which has a consequent (THEN clause) that matches a desired goal [55]. Searching strategies are very important for a rule based inference engine in terms of efficiency. For a complex system, there are a large number of rules and facts stored in the database to represent possible scenarios for the system, which results in a slow and computationally expensive reasoning system. For example, if a small system has 5 state variables, each variable has 5 states and needs to take different actions for different system states; there would have to be $4^5 = 1024$ rules in order to be exhaustive and complete. If the searching algorithm scans every individual rule and checks if the conditions are satisfied or not, the inference engine would be too

slow to be used in practice. The rule based engine is a well-developed inference method and there exist many commercial tools which can be used to develop a rule based inference engine for a particular system. For example, JESS is a rule-based engine scripting environment written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. JESS uses an enhanced version of the Rete algorithm to process rules. Rete was first designed by Dr. Charles L. Forgy of Carnegie Mellon University in 1974 and continues to be improved [56]. The Rete algorithm uses a rooted acyclic directed graph to store pattern information. It intends to improve the speed of forward-chained rule systems by limiting the effort required to recompute the conflicted set after a rule is fired.

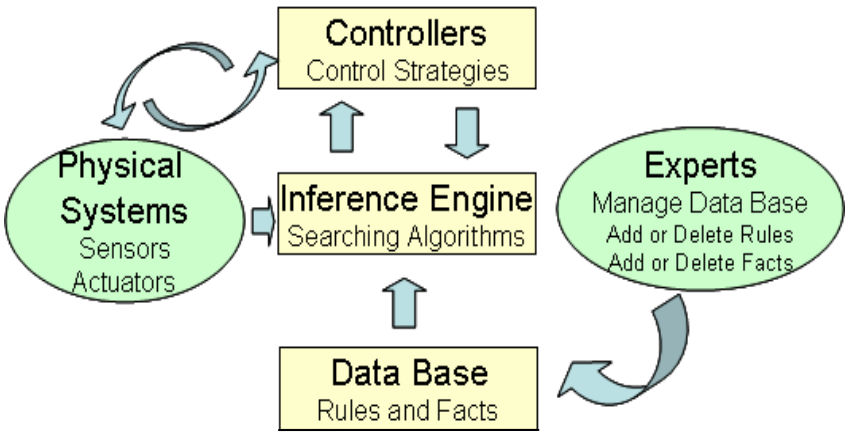


FIGURE 33 RULE-BASED INFERENCE ENGINE

A rule-based inference engine is not suitable for a complex system.

- First of all, as mentioned above, it is almost impossible to list all of the scenarios by using qualitative rules and facts to describe the characteristics of a complex system even when the system is static.
- Second, with a large number of rules and facts, searching the database efficiently is a very challenging task.

- Third, it is very hard to do accurate inference by using a limited set of rules when a complex system behavior is global in nature. By global, it means one part of the system states are not just dependent on particular components, but also related to some other components in the system, which is far away from the measured states in question [57]. The most common systems with global behaviors are fluid systems and electrical power systems. For example, for a chilled water cooling system, the flow rate at one particular service load is not just affected by the valve close to the measured point of this flow rate, but also affected by the pump rotation speed, other service load states, etc.
- Forth, a rule-based inference engine is hard to be distributed for a system with global behaviors. For a loosely coupled complex system, each subsystem can have its own relatively independent rule base and do the inference locally just by exchanging some facts information from other subsystems [58]. However, for a system with global behaviors, by using some local inferences, it is very challenging to keep globally consistent inferences. If a shared data base is used, it would face the aforementioned problems. And it will further slow down the inference process because of communication delays.
- Last, it is difficult to do inference under significant uncertainties by using a pure rule based inference engine.

An inference engine based on fuzzy logic is used to handle uncertain and imprecise information as an extension of expert inference reasoning method. Figure 34 shows the basic steps of fuzzy logic reasoning process:

- First, the system transforms crisp inputs into fuzzy inputs by using corresponding input set membership functions.
- Second, search the rule base and fire the matched rules.

- Third, combine the matched rules to get one normalized value.
- Last, defuzzify the normalized value back to the actual value according to corresponding output set membership functions.

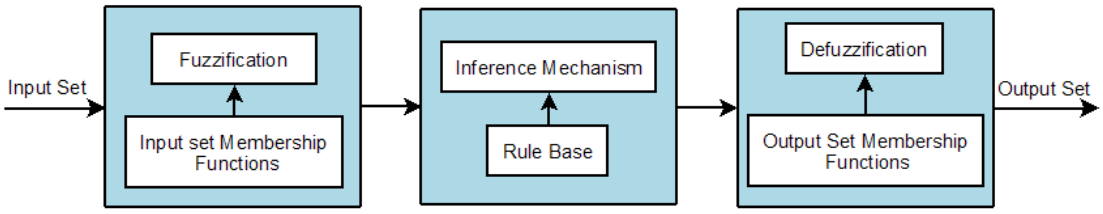


FIGURE 34 FUZZY LOGIC PROCESS

Perianu and Havinga introduced a distributed fuzzy logic engine for rule-based wireless sensor networks [59]. It is similar to the idea in [58] mentioned above except that the rules and inputs are fuzzificated to handle uncertain information. In summery, the fuzzy logic rule based inference engine as an extension of deterministic rule based inference engine shares the same problems for reasoning in large-scale complex system as the deterministic rule-based inference engine does.

4.4 Model-Based Inference Engine

The idea of model-based inference engine is simple: establish a model for the actual system and make inference of the actual system states by tuning the parameters of the model to match observations from the actual system. Figure 35 shows the structure of a model-based inference engine.

Model-based reasoning is an active research area and it has been implemented for some real systems. An exhaustive review of those implementations and their limitations are not necessary here. Most of those implementations focus on the searching strategies of the

model parameters to match the observations from the actual system and the outputs from the model. For example, McKenzie et al. applies a conflict-oriented approach to tune the parameters of a robust model of an electrical power system to match actual observations [57]. Maul et al. developed a diagnostic software package which provides diagnostic information about the operational condition of the modeled rocket engine system or subsystems [60]. The diagnostic package uses a constraint suspension algorithm to direct qualitative model solver operations to provide valuable information about the modeled system. However, constraint suspension algorithm is difficult or impossible to be implemented for a complex system with global behavior, because constraint suspension algorithm uses constraints for localized components corresponding to the abnormal observations. Williams and Nayak described Livingstone, an implemented kernel for a self-reconfiguring autonomous system [61]. Liveingstone uses component-based declarative models. Karp et al. gave a detailed review of model-based reasoning and applied model-based reasoning for electro-mechanical devices [62].

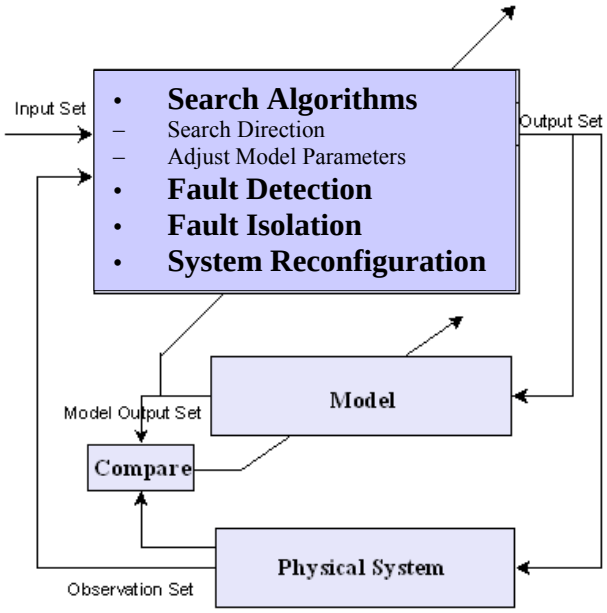


FIGURE 35 MODEL-BASED INFERENCE ENGINE

State estimation by using a structure-behavior model simulating the actual system dynamic characteristics eliminates the problem of listing huge set of system scenarios and the problem of unforeseen situations, because all of the scenarios are interweaved into the structured model and can be mimicked by changing the inputs for the model at run time. The complexity of the inference involved would not grow based on the patterns of the system states but rather would be based on the number of components in the system/the complexity of the system itself [57].

However, the accuracy of model-based reasoning highly depends on how much the model captures the actual system dynamic characteristics and how robust the model behaves. A rough model or a model being sensitive to numerical noises will degrade the accuracy of the inferences dramatically. Establishing a global consistent model for a complex system is a very challenging task and may not be practical. Even such a model exists, but how to control the model parameters to match the observations from the actual physical system is still an active research field and there is no general method for tuning the model parameters to efficiently decrease the discrepancy between the outputs from the model and the observations from the actual system. More importantly, if the model itself is complicated enough, even one run for such a model may take a long time to finish, thus it will further slow down the reasoning process.

For a distributed model-based reasoning, one agent is constructed with a single model of a subsystem being controlled by the agent, a single set of feasible states and a single search engine. Each agent uses its own relatively independent search engine to manage the generation of conflicts and perform diagnosis of its controlled subsystem. Coordination between agents is facilitated through an agent messaging system [22]. However, in [22], it does not mention how the agents coordinate with each other and how to deal with uncertainties.

4.5 General Probabilistic Networks

4.5.1 Sensor Networks from a Multi-Agent Perspective

In this section, a brief discussion of sensor networks from a multi-agent perspective is given to show the suitability of implementation of multi-agent system in sensor networks for state estimation under noise contaminated observations, drawing from [63, 64] and references therein.

Recent advancement in communication and micro-electronic techniques has enabled the development of wide-distributed and low-cost sensor networks. Such networks consist of multiple sensors, deployed over a wide area, connected through a communication network (wired or otherwise). Sensor networks and intelligent arrays of micro-sensors have broad range of applications including information gathering and data fusion for modeling an environment, surveillance, active monitoring of forests & agriculture lands, health-care applications[18-22].

Sensor with embedded small processor can be appropriately modeled as an autonomous self-aware agent in a flexible way[64]. In sensor networks, sensor agents may go beyond reacting to their local situations, transferring raw data set or simply data preprocessing. While many sensors are distributed in a wide-range area and connected through wired/wireless communication networks, it is called Distributed Sensor Networks (DSNs). Coordination among different sensors/agents falls exactly into multi-agent collaboration categories. DSNs as a revolutionary new technology, it is an on-going research and involves many techniques, such as wireless networks, signal processing, self-organizing and multi-entities coordination, etc. The resource of complete and consistent introduction of DSNs is scarce and most related information is scattered in different papers. Iyengar and Brooks collected and edited over 500 illustrations and over 1000 pages of in depth

information can be used as a good starting point and reference source[65] for studying DSNs.

This dissertation will focus on the aspects of self-organizing and multi-entities coordination for system state reasoning with uncertainty; wireless networks, signal processing techniques and optimal sensor allocations will not be further discussed. Distributed Kalman Filtering (DKF)[66-71], Distributed Particle Filters (DPFs)[72-76] and Distributed Dynamic Bayesian Networks (DDBNs)[49, 77-80] are three widely researched and implemented state reasoning methods in DSNs.

4.5.2 Distributed Kalman Filtering (DKF)

DKF is one of the most fundamental distributed estimation methods of scalable sensor data fusion for linear dynamic systems with Gaussian noise distributions. Assume the system with linear dynamics:

$$x_{k+1} = A_k x_k + B_k w_k \text{ ; given } x_0 \tag{4.2}$$

where, $x_k \in \mathbb{R}^n$, $A_k \in \mathbb{R}^{n \times n}$, $x_0 \in \mathbb{R}^n$, $w_k \sim N(0, Q)$ represents the process noise and $x_0 \sim N(\bar{x}_0, P_0)$ indicates the initial system state distribution.

Now, assume there are m sub sensor networks with the corresponding observation models:

$$z_i(k) = H_i x(k) + v_i(k) \text{ , } i = 1, 2, \cdots, m \tag{4.3}$$

where, $z_k \in \mathbb{R}^l$, $H_k \in \mathbb{R}^{n \times l}$, $v_i(k) \sim N(0, R_i)$ represent observation noises and are independent.

$$S = \frac{1}{n} \sum_{i=1}^{m} H_i' R_i^{-1} H_i \tag{4.4}$$

$$y_i = H_i' R_i^{-1} z_i \tag{4.5}$$

$$y = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{4.6}$$

Distributed estimation in DKF is solved by calculating averaging inverse covariance S and average measurements y in a distributed way for each node at every iteration. Low-pass consensus filter and band-pass consensus filter are used to calculate average measurements y and averaging inverse covariance S through sub sensor node communications involving only neighbor nodes [66, 67, 81, 82]. DKF is suitable for linear dynamic system with Gaussian distributed noises. In non-Gaussian and non-linear situations, Extended Distributed Kalman Filters (EDKFs), grid-based methods and Gaussian-sum filters are possible alternatives, but these methods have limited applications and coordination among different entities with limited communications is a very challenging issue [66, 83, 84].

4.5.3 Distributed Particle Filters (DPFs)

DPFs is another attractive distributed state estimation method due to its power and flexibility. The following sections briefly outlines DPFs, drawing on descriptions from [72, 73] and references therein.

DPFs consists of a set of particles. Each particle presents a state trajectory and a corresponding series of weights indicating how well this state trajectory conforms to the dynamic models and explains the available observations at each time point. It is not desirable to transfer raw data to a set of processing data nodes in DPFs. Despite the advances in silicon fabrication technologies, data transmission and idle listening is and will continue to be more time and energy consuming than data processing for the foreseeable future, especially for communication over the wireless networks [85] [86]. Therefore, what information to be transferred and how to reach globally consistent

estimations with limited communications between different nodes is the most challenging part of DPFs design for DSNs. Currently, there are two general types of algorithms: parametric approximation and adaptive data-encoding. The first algorithm is based on the establishment of parametric models for the distribution and only transfers the model parameters among different nodes. The second algorithm uses an adaptive data-encoding approach. It compresses the data set and only transfers the most informative measurement (observations) to its neighbors. By using this method, it is important to evaluate information of a measurement, which may involve more query-answer communications between different nodes. Most particle filtering algorithms are based on two assumptions: State Markov Property (SMP) and Observation Markov Property (OMP). These two assumptions are the same as the hidden variable Markov property and observable variable Markov property in a two-time slice homogeneous Dynamic Bayesian Network (DBN) described in the next section.

- State Markov Property: $P(x_t \mid x_0, \cdots, x_{t-1}) = P(x_t \mid x_{t-1})$. It says $x_0, x_1, \cdots$ is a first order Markov process.
- Observation Markov Property: $P(y_t \mid x_0, \cdots, x_{t-1}, x_t, y_0, \cdots, y_{t-1},) = P(y_t \mid x_t)$. It says the observations $y_0, y_1, \cdots$ are conditionally independent provided that $x_0, x_1, \cdots$ are known.

Both DKF and DPFs methods are suitable for evaluation of the whole system state with distributed observations at each node. However, in most of situations, it is not necessary for each node to know everything about the system. Each local node only needs partial information of the whole system, which will save a lot of time for data processing and data communication. DDBNs is a distributed Bayesian network for dynamic systems. DBN has been well studied and implemented in different applications, such as network

security measuring and monitoring[87], speech recognition[88], supply chain diagnostics[89], influenza surveillance[90], etc.

In Chapter III, a preliminary introduction on Bayesian networks has given. However, before introducing DDBNs as a distributed dynamic probabilistic inference engine, two more topics on how a Bayesian network deals with dynamic systems and how to handle directed cycles in a monolithic Bayesian network respectively are discussed first.

4.5.4 Dynamic Bayesian Networks

4.5.4.1 Introduction

Most large-scale complex systems have dynamic characteristics. Observations obtained through sensors are time series. States of components are changing over time. For a real-time control system, the system states are required at every time for controllers to make right decisions. Usually, component states at difference time slices are correlated. How to model variable relationships at different time slices? It is natural to use directed graphical models which can capture the fact that time flows forward. Arcs within a time-slice can be directed or undirected. If both of arcs within one time slice and arcs between two time slices are directed, such a model is called a DBN [91]. Here, the term "dynamic" means the system states are evolving over time and it is nothing to do with the system structure at each local time slice.

DBN's structure at each local time slice can be fixed or changed over time. If all of the parameters (It includes both of the cause-effect relationships and the conditional probability distributions) at different local time slices are identical, such a model is a repetitive temporal model. In this dissertation, we will focus on DBN with fixed structures. Another question for a DBN is how many time slices will be connected directly. As we know, the complexity of a Bayesian network grows exponentially with the number of nodes. If too many time slices are connected directly, the Bayesian

network will easily grow too large to handle. Fortunately, the two-time slice Markov Property based DBN works well in many practical situations. Therefore, in this research, a homogenous two-time slice DBN will be used.

Divide the nodes in a homogeneous two-time slice DBN into two sets: unobserved states W and observed states E. It includes three main elements:

- a prior model,
- a transition model,
- an observation model.

A two-time slice homogeneous DBN is shown in Figure 36, compared with a general DBN as shown in Figure 37.

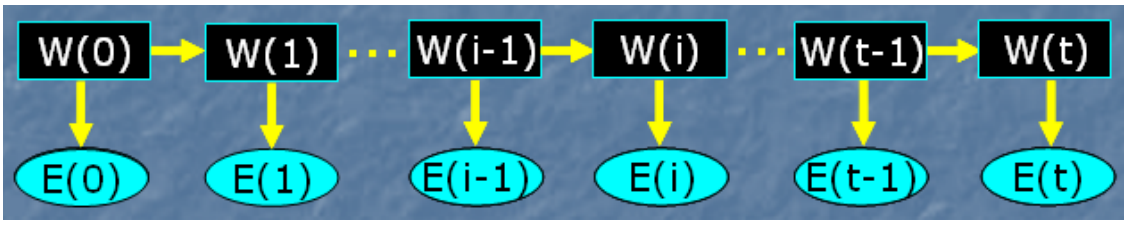


FIGURE 36 TWO-TIME SLICE HOMOGENOUS DYNAMIC BAYESIAN NETWORKS

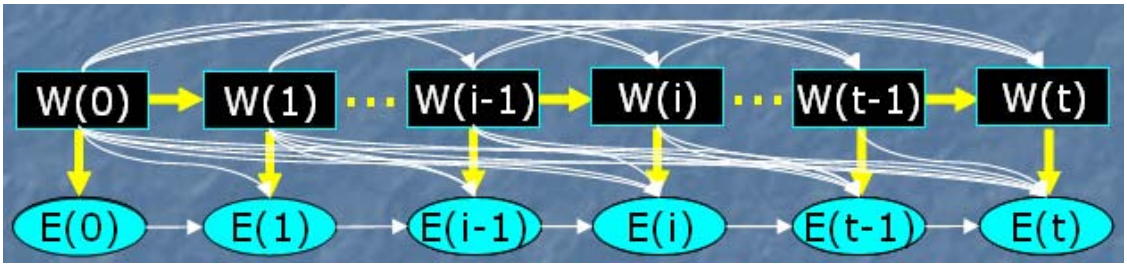


FIGURE 37 A GENERAL DYNAMIC BAYESIAN NETWORK

A prior model is a cause-effect model in the same time slice for the hidden variables. It captures the relationships of a dynamic system in one time slice. Such a prior model

domain is $W(t)$. Here, $W(t)$ is a vector. The conditional probability distributions are like $P(w_i(t) \mid w_k(t), \cdots, w_m(t))$.

A transition model is a model describing the relationships between two time slices. Such a transition model domain is $(W(t), W(t+1))$. The conditional probability distribution can be shown as $P(W(t+1) \mid W(t))$.

An observation model is a model representing the relationships between the hidden variables and the observable variables in one time slice. The condition probability distribution can be shown as $P(E(t) \mid W(t))$.

In a homogeneous two-time slice DBN, three assumptions are made:

- Time-invariant/homogeneous: the cause-effect relationships and the conditional distributions in time slice t_i is as the same as the cause-effect relationship and the conditional distributions at time slice t_j, for $\forall i \neq j$.
- Hidden variables Markov property: $P(W_t \mid W_0, \cdots, W_{t-1}) = P(W_t \mid W_{t-1})$. It says the hidden variable distributions at time t are conditionally independent of previous hidden variable distributions from time 0 to time $t-2$ if the hidden variables at time $t-1$ are specified.
- Observable variables Markov property: $P(E_t \mid W_0, \cdots, W_{t-1}, E_0, \cdots, E_{t-1},) = P(E_t \mid W_{t-1})$. It says the observable variable distributions at time t are conditionally independent of previous hidden variables from time 0 to time $t-2$ and previous observable variables from time 0 to time t-1 if the hidden variables at time $t-1$ are specified.

These three models are used to define the joint distribution for the DBN up to any fixed time T, which is given by

$$P_{V(1:T)} = P_{W(0)} \prod_{t=1}^{T} P_{W(t)|W(t-1)} \prod_{t=1}^{T} P_{E(t)|W(t)} \tag{4.7}$$

Forming this joint distribution for a given time T is called unrolling the DBN to time T [92]. $P_{V(1:T)}$ can be integrated to show joint distributions of different sets of state variables at different times for different usage. There are three types of usage of a DBN:

- prediction, $P_{W(T+\tau)|E(0:T)}, \tau > 0$,
- smoothing, $P_{W(T-\tau)|E(0:T)}, \tau > 0$,
- filtering, $P_{W(T)|E(0:T)}$.

In this research, the DBN is only used as a filter to do state estimation: $P_{W(T)|E(0:T)}$, which can be further simplified as follows:

$$P_{W(T)|E(0:T)} \propto P_{W(T)|E(0:T-1)} P_{E(T)|W(T)} \propto P_{W(T-1)|E(0:T-1)} P_{W(T)|W(T-1)} P_{E(T)|W(T)} \tag{4.8}$$

The system does not need to store all of $P_{W(T-1)|E(0:T-1)}$s from time 0 to time $T-1$. It is an iterative process and only the current $P_{W(T-1)|E(0:T-1)}$ needs to be stored in memory for next time state estimation.

4.5.4.2 Belief Updating for Two-Time Slice Homogenous Dynamic Bayesian Networks

Belief updating for general DBNs is a very challenging job. As we know, an inference in general Bayesian Networks is NP-hard. Unrolling the whole DBN into one flat Bayesian Network, the number of nodes is linearly increasing with time slices. As discussed in the previous section, two-time slice homogenous DBN is most commonly used in practical situations. Here, drawn from Murphy's dissertation [93] and its corresponding references,

an introduction to two belief updating algorithms for two-time slice homogenous DBNs are given: frontier algorithm and interface algorithm.

As we know, $P_{W(T-1)|E(0:T-1)}P_{W(T)|W(T-1)}P_{E(T)|W(T)}$ contains multiplication of $P_{W(T)|W(T-1)}$. Assume $W(T)\in\mathbb{R}^n$ and $w_i(T)$ has m discrete states, then the joint conditional distribution of $P_{W(T)|W(T-1)}$ is a $\mathrm{O}\left(m^n\times m^n\right)$ matrix, which is big even for a small problem. The basic idea of frontier algorithm is to updating $P_{W(T)|E0:T)}$ by "delete/add" nodes one by one from/to a frontier set (the frontier set d-separate the nodes in the left side and the nodes in the right side) across the DBN instead of multiplying the $\mathrm{O}\left(m^n\times m^n\right)$ transition matrix. It includes two steps: forwards pass and backwards pass for data smoothing. For forwards pass, add a node into the frontier set when all its parents are already in the frontier set; remove a node from the frontier set when all its children are in the frontier set. Forwards pass advances from left side (time slice T) to right side (time slice $T+1$). For backwards pass, remove a node from the frontier set when all its parents are already in the frontier set; add a node from the frontier set when all its children are in the frontier set. Backwards pass advances from right side (time slice $T+1$) to left side (time slice T). Forwards pass is followed by backwards pass. Frontier Algorithm is for data smoothing. In the application of this dissertation, only on-line state estimation (data filtering) is needed for the control system and backwards pass can be neglected.

In the frontier algorithm, the frontier set contains all the hidden nodes in a time slice. This set is larger that it needs to be. As we know, the frontier set is the set which can d-separate the past from the future. The set of nodes with outgoing arcs to the next time-slice is sufficient to d-separate the past from the future and this set is called "forward interface" by Murphy[93]. Interface algorithm also uses forwards pass and backwards pass which are similar to the frontier algorithm.

Both of frontier algorithm and interface algorithm are exact inferences for discrete state-spaces problems. They are possible, but maybe computationally prohibitive and make exact inference intractable for large discrete DBNs.

Boyen-Koller (BK) algorithm is a standard approximation inference in discrete-state problems. BK approximates the joint distribution over the interface as a product of marginals. The basic idea of BK is to construct the junction tree for the $1\frac{1}{2}$ time slices for a DBN without requiring all the interface nodes to be in the same clique. The accuracy of the BK algorithm depends on the clusters used to approximate the belief state of the interface set. An extreme usage of BK algorithm is to completely decompose the nodes of the interface set into small clusters as one cluster contains one variable.

4.5.5 Directed Cycles in Graphical Models

As we know, Bayesian networks are directed acyclic graphical models. Bayesian networks can not handle directed cycles in the model. This makes sense because Bayesian networks are cause-effect models. If there is a directed cycle, the effect becomes the cause and the inference process will be stuck into an infinite loop or some conflicts. For example, V_1 is the parent of V_2; V_2 is the parent of V_3 and V_3 is the parent of V_1. This forms a directed cycle as shown in Figure 38.

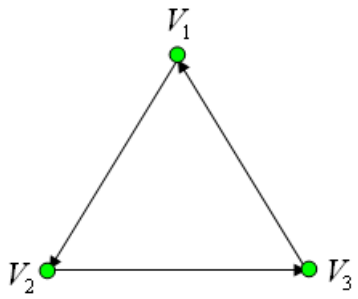


FIGURE 38 A DIRECTED CYCLE

The potentials for this cycle are $P(V_2 \mid V_1)$, $P(V_3 \mid V_2)$ and $P(V_1 \mid V_3)$. Assume V_1, V_2 and V_3 have two states 0 and 1 respectively and the three conditional probabilities are the same for the sake of convenience. The potentials are shown in Table 2. Now, assume that evidence shows $V_1 = 1$. By exploiting the structure of this network, we get $V_2 = 0$, $V_3 = 1$ and in turn $V_1 = 0$, which is a conflict with the observation $V_1 = 1$.

TABLE 2 CONDITIONAL POTENTIAL TABLE

Child / Parent / Con Prob	0	1
0	0	1
1	1	0

Cycles will not appear when Bayesian networks are established for most of physical systems except for some special cases, such as a recycled chilled water system shown in Figure 39, if flow rate at different points is a node in the Bayesian network. In this simple recycled chilled water system illustrated in Figure 39 , there are 6 controllable components (two pumps and 6 valves), which can affect the flow rate in the system.

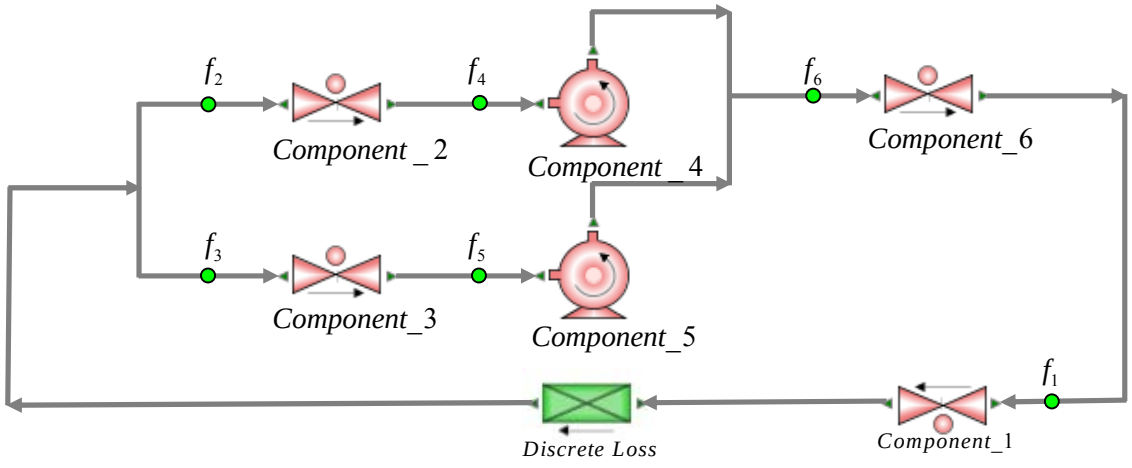


FIGURE 39 A RECYCLED CHILLED WATER SYSTEM

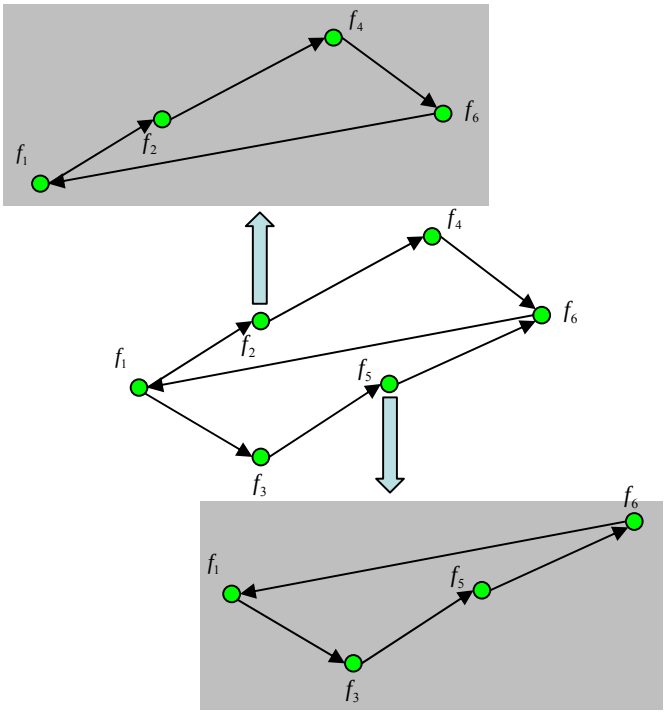


FIGURE 40 CYCLES IN THE BAYESIAN NETWORK FOR THE RECYCLED CHILLED WATER SYSTEM EXAMPLE

When we use the flow rates measured at 6 different locations as nodes of a Bayesian network, two cycles connecting these 6 nodes will be formed naturally. Those cycles are shown in Figure 40.

As mentioned before, Bayesian Networks can not handle cycles, so we need a way to break those cycles but still keep the system characteristics and perform consistent inference reasoning.

The first method is to break the directed cycle at one point. For example, if the edge connecting f_1 and f_6 is deleted, the two directed cycles shown in Figure 40 are broken and the results are shown in Figure 41. However, by doing that, the information between f_1 and f_6 is lost.

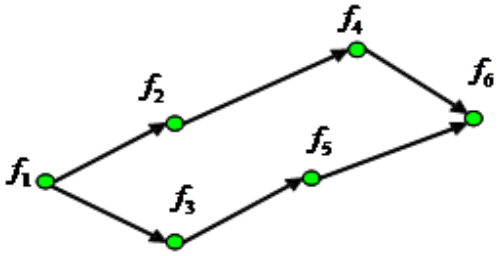


FIGURE 41 BREAKING DIRECTED CYCLES IN RECYCLED FLUID SYSTEM BAYESIAN NETWORK

The second method is to change the direction of one edge in a directed cycle if the directions of some edges are not very important. For example, if the direction of edge between f_1 and f_6 is changed from f_1 to f_6, there is no directed cycle any more and the result is shown in Figure 42. This method is simple and does not complicate the existed Bayesian network. However, this method is based on the assumption that cause-effect relationship is not significant between two nodes and is limited to certain applications.

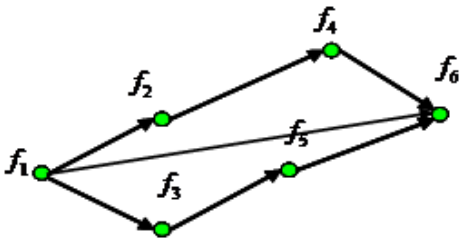


FIGURE 42 CHANGING DIRECTION OF f_1 AND f_6 IN RECYCLED FLUID SYSTEM BAYESIAN NETWORK

The third method is to add an intervention (an instantiation of a node) to break the cycle [94]. As we know, in the sum-product algorithm for discrete Bayesian network inference, when a node V_i is explicitly instantiated to a specific value, the conditional probability function $P(V_i \mid parent(V_i))$ will be removed from the sum-product equations. Viewing

this removing action graphically, the edges into this specifically instantiated node are deleted from the graphical model. For example, in the simple chilled water system Bayesian network, if node f_1 is instantiated to a specific value, the mass probability function $P(f_1 \mid f_6)$ will be removed from the sum-product equation: $P(f_1, f_2, f_3, f_4, f_5, f_6) = P(f_1 \mid f_6) \times P(f_2 \mid f_1) \times P(f_4 \mid f_2) \times P(f_6 \mid f_4, f_5) \times P(f_3 \mid f_1) \times P(f_5 \mid f_3)$. Viewing that removal graphically, the directed edge from f_6 to f_1 be removed and the resulted graph is shown in Figure 43. Compared with the second method, the resulted graph looks as the same. However, the third method does not lose any information and it requires f_1 being instantiated. If f_1 can not be instantiated, other nodes in the cycle can be candidates to be instantiated and will break the cycle as well. If no node can be instantiated (observed) in a directed cycle, the third method can not be used. By using the third method, it simplifies the model and keeps the system internal cause-effect relationships. Herein, it will be used in the applications of this dissertation.

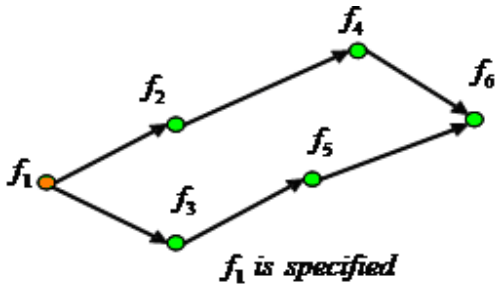


FIGURE 43 BREAK A DIRECTED CYCLE BY INTERVENTION

4.5.6 Hypothesis 2.1

Now, we know how Bayesian networks work for a dynamic and uncertain system, a hypothesis corresponding to research question 2.1 follows naturally:

- ***H2.1: If Dynamic Bayesian Networks (DBNs) can be established by investigating cause-effect relationships among different variables, incomplete and uncertain information can be handled systematically and efficiently.***

4.6 Distributed Bayesian Networks

4.6.1 General Ideas of Distributed Bayesian Networks

"*Instead of propagating all of the information everywhere, it is possible to assess first the potential impact of every updating operation on the belief of the target node and to limit the updating process so that only relevant information is propagated. Doing so will decrease the amount of data traffic in the network and the amount of computation expended on interference. However, it is important that the information we choose not to propagate be allowed to accumulate at the boundaries and discharge its impact to new areas of knowledge once our current set of belief becomes stagnant.*" [95]

-- Pearl

Pearl expressed a significant idea about distributed inference engine concisely at a conceptual level. Basically, similar to the discussion in previous sections, a distributed Bayesian network for state inference includes four aspects as a general distributed inference engine does:

- local information processing,
- partial intermediate information exchange,
- inference global consistency,
- self-organization due to partial damage.

Each agent represents its knowledge in a sub Bayesian network. Reasonable and working distributed Bayesian networks should have the following basic capabilities as a general distributed inference engine does:

- If an agent is isolated from other agents, it can infer its local node states based on its local available information correctly (Local Intelligence and Independence).
- If some links between two agents are damaged, an agent can infer its local nodes based on its local measurements and the available messages it receives from its neighbors (Local Robustness and Optimization).
- If all of the connections among the agents are not damaged, through communication, the state estimations for all of the agents are consistent (Coordination and System Consistency).
- If some agents are missing or some links between two agents are damaged, the remaining agents can reorganize themselves into certain structures based on existed link quality. Coordinated inferences can be initiated through the reorganized structures (Self-organization and Inference Automation).

Currently, there exist three types of distributed Bayesian networks: Distributed Perception Networks (DPNs), Prior/likelihood Decomposable Models (PDM) and Multiple Sectioned Bayesian Networks (MSBNs). All of them provide frameworks with different algorithms to partially implement such a conceptual idea of distributed inference engine. Before discussing the three distributed Bayesian networks in detail, a few concepts are introduced first.

4.6.2 Basic Concepts in Distributed Bayesian Networks

Graph Consistent: two graphs G_i and G_j, and let $S_{ij} = dom(G_i) \cap dom(G_j)$. We say G_i and G_j are graph consistent if the sub graph of G_i spanned on S_{ij} is identical to the sub graph of G_j spanned on S_{ij}.

Two consistent graphs can be united to form one unique graph and can be partitioned back to the two original graphs without modifying any edges or nodes.

D-sep Set: let $G_i = (N_i, E_i)$ and $G_j = (N_j, E_j)$ be two consistent DAGs, such that $G = G_i \cup G_j$ is a DAG. The intersection set $S_{ij} = G_i \cap G_j$ is a d-sep set between G_i and G_j if and only if for every $V \in S_{ij}$ with all of its parents set $Pa(V)$, either $Pa(V) \subseteq N_i$ or $Pa(V) \subseteq N_j$. Each $V \in S_{ij}$ is called a d-sep node.

Hyper Tree: Let $G = (N, E)$ be a connected graph decomposed into a set of sub graphs $\{G_i = (N_i, E_i)\}$, $i = 1, \cdots n$, where $G = \bigcup_{i=1,\cdots n} G_i$. Let the sub graph set $\{G_i = (N_i, E_i)\}$, $i = 1, \cdots n$ be organized into a undirected tree ψ (there is a unique path between two nodes in tree ψ) where each node is uniquely labeled by G_i and each link between G_i and G_j is labeled by the nonempty intersection set $S_{ij} = G_i \cap G_j$, such that for each i and j, S_{ij} is contained in each sub graph on the path between G_i and G_j in ψ (hyper tree satisfies the running intersection property as defined in Chapter II), then ψ is a hyper tree over G. Each G_i is a hyper node and each intersection set S_{ij} is a hyperlink.

A Hyper tree Multiple Section Directed Acyclic Graph (Hyper tree MSDAG) is $G = \bigcup_{i=1,\cdots n} G_i$, where each G_i is a DAG. For two neighbors G_i and G_j, they are graphically consistent. G is a connected DAG and it satisfy two conditions:

- $\exists$ a hyper tree ψ over the set of $\{G_i = (N_i, E_i)\}$.
- Each hyperlink in ψ is a d-sep set.

4.6.2.1 Justification of Tree Organization in Distributed Bayesian Networks

As we know, in order to make coherent and consistent inferences over distributed Bayesian networks, the organization of the set of distributed Bayesian networks should satisfy the tree structure condition with running intersection property (hyper tree). Xiang and Lesser justified the necessity of the tree structure by introducing two types of loops existing in common distributed Bayesian networks [96].

Let G be a cluster graph with domain V and ρ be a loop in G. If there exists a separator S on ρ contained in every other separator on ρ, then ρ is a degenerate loop. Otherwise, ρ is a non-degenerate loop.

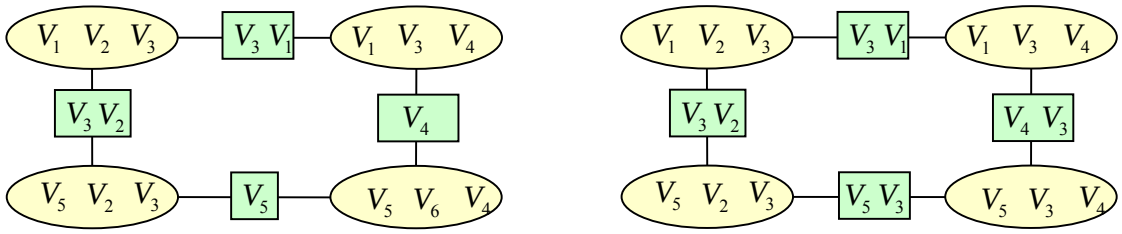
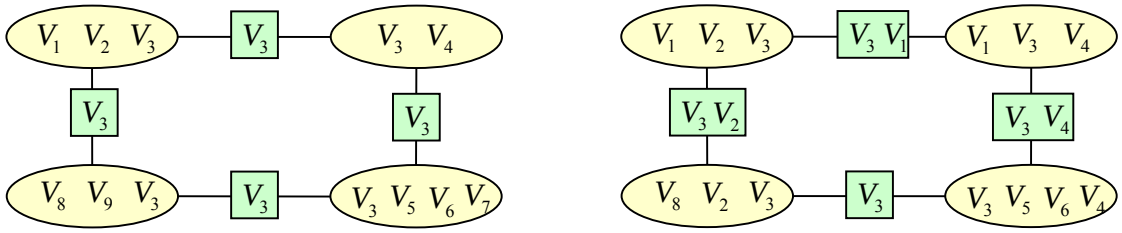


FIGURE 44 NON-DEGENERATE LOOP



(1) Strong Degenerate Loop (2) Weak Degenerate Loop

FIGURE 45 DEGENERATE LOOP

Degenerate loop can be further divided into two sub-types: ρ is a strong degenerate loop where all separators on the loop ρ are identical; otherwise, ρ is a weak degenerate loop. Figure 44 shows two non-degenerate loops. Figure 45 shows a strong degenerate loop and a weak degenerate loop respectively.

For non-degenerate loops, messages transferred between different clusters in the loop can not become informative and inferences can not be made coherent and consistent, no matter how the messages are communicated among the clusters in the loop [96].

For a degenerate loop, there are two ways to transfer the information from one point to another point and one of them is redundant. Therefore whether or not coherent message passing is achievable in a weak degenerate loop depends on the cluster chain obtained by breaking the loop at the separator with the smallest domain set [96]. If the resulted chain satisfies the running property, then the message communications will be formative and consistent, otherwise, it would not be consistent in general. Furthermore, the resulted chain will make the same inferences as the loop does but in a more simple and direct way.

Based on the above two reasons, a tree organization with running property is the simplest, most efficient structure for a distributed Bayesian network to make coherent message passing and consistent inferences.

As well as a tree organization with running intersection property is necessary for consistent inference, d-sep set condition is necessary for conditionally independence between two neighbored sub graphs given their separator information. In summary, for a distributed Bayesian network, in order to make globally consistent inferences in the simplest and distributed way through message passing over the shared variables between two neighbored sub networks, there are three requirements:

- All of the sub Bayesian networks are organized into a tree structure.

- The tree structure satisfies the running intersection property.
- Each node shared by two or more sub Bayesian networks should be a d-sep node.

4.6.3 Distributed Perception Network

DPNs is a distributed architecture for efficient and reliable fusion of large quantities of heterogeneous and noisy information [97]. DPNs is composed of numerous agents who cooperate with each other to process systematic reasoning. However, DPNs has a few restrictions which limit its application region.

DPNs Domain Model: a DPNs domain model M is defined as a tuple $D=(G,V,S,P)$, where G is the set of local DAGs of all DPNs agents participating in a particular fusion organization; V is a union of all variables from local clusters and $Q_h \in V$ that contains the hypothesis node; $S=\{S_1,S_2,\cdots,S_{m-1},S_m\}$ is the set of DPNs separators and every separator $S_i \in S$ has to be unique and satisfy restriction 1(see below); P is the set of potentials defined over the DPNs domain model and $P_i \in P$ are the potentials for cluster C_i, where $C_i \in T$ and $T=\{T_1,T_2,\cdots,T_{n-1},T_n\}$ satisfying a tree architecture and the running intersection property.

A local DAG G_i with domain Q_i in a DPNs domain model contains a single root node corresponding to a service variable $R \in Q_i$ and a set of input variables corresponding to a set of leaf nodes $L_i \in Q_i$. The leaf nodes are always the descendants of the service node.

DPNs has very strict organization constraints which make its implementation limited to a few situations.

- Restriction 1: for any $S_i \in S$, it contains only one unique variable.

- Restriction 2: when adding a new agent with its cluster domain C_i, it can connect to only one unique $C_j \in T$ with a separator S_k, where S_k contains only one variable.
- Restriction 3: two local DAGs G_i and G_j with domains Q_i and Q_j respectively can connect each other if and only if the service variable $R_i \in Q_i$ of G_i is identical to an input variable $R_i \in L_j \subset Q_j$, where L_j is the input variable set of G_j.

If one agent joins the system, it should satisfy these three restrictions at the same time. These three constraints are too strict. For a realistic system, normally, one agent could connect to several agents and an interface (separator set) between two agents contains several variables. A local DAG G_i may contain several root nodes. A node in a separator could be an input variable of one local DAG as well as an input variable of another local DAG.

DPNs deals with three types of agents: static modeling agents, dynamic modeling agents, and appendable modeling agents. Each agent type updates its belief by using a specific algorithm.

Static Modeling Agents

An agent who implements static modeling building blocks can reason in an integrated way about distributions over some quasi static variables. An event is quasi static if it does not change before the resulting observations are interpreted and used in a decision making process. In another word, a quasi static event does not involve for a certain period of time after they have been materialized. For example, if a cow is infected, the cow would not be cured during one time step of information fusion process. An algorithm for static fusion process is shown in Figure 46.

procedure: StaticFusion $(\phi'(S\langle Q_t, Q_l\rangle))$
input : Separator potential $\phi'(S\langle Q_t, Q_l\rangle)$ over a leaf node (corresponding to soft evidence)
output : Distribution $P(R_s)$ over the service root

1 Update potential $\phi(Q_t)$ of graph G_t with the received separator potential $\phi(S'\langle Q_t, Q_l\rangle)$ from agent A_l with variables Q_l with:

$$\phi(Q_t) = \frac{\phi(Q_t)\phi'(S\langle Q_t, Q_l\rangle)}{\phi(S\langle Q_t, Q_l\rangle)}$$

where $\phi(S\langle Q_t, Q_l\rangle)$ is the old separator potential;

2 Set $\phi(S\langle Q_t, Q_l\rangle) \leftarrow \phi'(S\langle Q_t, Q_l\rangle)$;

3 compute $\phi(R_s) = \phi(Q_t)^{\downarrow R_s}$;

4 Return $P(R_s) = \alpha \cdot \phi(R_s)$;

FIGURE 46 STATIC FUSION ALGORITHM [97]

Dynamic Modeling Agents

An agent who implements dynamic modeling building blocks can infer from a time series observations. An algorithm for dynamic fusion process is shown in Figure 47. This algorithm has to satisfy two assumptions:

- All observations are conditionally independent given the sensor propensity.
- The generative model is the same for all observations of a certain type. It means that all of the observations for the same sensor at different time steps are sampled by using the same method and with the same model.

procedure: DynamicFusion $(\phi(e_{E_k}))$
input : Hard evidence encoded in $\phi(e_{E_k})$
output : Distribution $P(R_s)$ over the service root

1 $\phi(R) = P(R_s)P(E_k|R_s)\phi(e_{E_k})$;

2 $P(R) \leftarrow \alpha \cdot \phi(R)$;

3 Return $P(R_s)$;

FIGURE 47 DYNAMIC FUSION ALGORITHM [97]

Appendable Modeling Components

An agent implements an additional modeling building block. Such an agent is used to support extensibility for the multi-agent system and makes it flexible and scalable. An algorithm for appendable fusion process is proposed in [97].

All of the agents need to collaborate together to achieve a reasoning task which maps available observations to some hypotheses. How to cooperate smoothly and efficiently? Two algorithms are used corresponding to two different situations.

procedure: AppendableFusion ($\phi'(S\langle Q_i, Q_k\rangle)$)
input : Separator potential $\phi'(S\langle Q_i, Q_k\rangle)$ over a leaf node (corresponding to soft evidence)
output : Distribution $P(R_s)$ over the service root

1

$$\phi'_k(R_s) \leftarrow \left(P(R_s) \sum_{L_k} P(L_k|R_s)\phi'(S\langle Q_i, Q_k\rangle) \right)^{\downarrow R_s}$$

if $S\langle Q_i, Q_k\rangle$ *is from a new information source* **then**
2 $\phi_\Psi(R_s) \leftarrow \phi_\Psi(R_s)\phi'_k(R_s)$;
3 Append uniform potential $\phi_k(R_s)$ to a list of separator potentials of all leaf nodes;
4 **else**
5 $\phi_\Psi(R_s) \leftarrow \frac{\phi_\Psi(R_s)\phi'_k(R_s)}{\phi_k(R_s)}$;
6 **end**
7 Set $\phi_k(R_s) \leftarrow \phi'_k(R_s)$;
8 Return $P(R_s) = \alpha \cdot \phi_\Psi(R_s)$;

FIGURE 48 APPENDABLE FUSION ALGORITHM [97]

Algorithm 1: Top down network configuration

This algorithm is used for self-organization based on the collaboration among multiple agents in a distributed way without any centralized and dominated control agent in order to answer a query of a unique service variable. This algorithm implements a simple self-organization rule: when a query of a service variable is made, it searches the agents

containing this service variable. For each of the matched agents, it starts to look for other agents who connect to it by its leaf nodes and satisfy restriction 3. After that, a set of DPNs are formulated, and then use an algorithm similar to collect evidence process introduced in Chapter 2 to calculate the potential of the queried service variable. In the collect evidence process, different agents will use their corresponding algorithms to update their own believes.

procedure: `TopDownConfiguration` (X)
input : X is a query concept

1 Find a set of agents $\mathcal{A}_q \subseteq \mathcal{A}$ such that $\forall A_i \in \mathcal{A}_q : R_i = X$;
2 **foreach** *agent* $A_i \in \mathcal{A}_q$ **do**
3 Use CNET Protocol to establish a fusion contract with A_i by satisfying the organization constraints in definition 4.6;
4 **if** *fusion contract with* A_i *is established successfully* **then**
5 **foreach** *input concept* $L_{i,j} \in \mathcal{L}_i$ *from agent* A_i **do**
6 in agent A_i call `TopDownConfiguration` ($L_{i,j}$);
7 **end**
8 **end**
9 **end**

FIGURE 49 TOP-DOWN CONFIGURATION ALGORITHM [78]

Algorithm 2: Bottom-up Network Configuration

procedure: `BottomUpConfiguration` (R_k)
input : R_k is the service concept of the caller agent A_k

1 Find a set of agents $\mathcal{A}_s \subseteq \mathcal{A}$ such that $\forall A_i \in \mathcal{A}_s : R_k \in \mathcal{L}_i$;
2 **foreach** *agent* $A_i \in \mathcal{A}_s$ **do**
3 Use CNET Protocol to establish a fusion contract between the caller agent and A_i by satisfying the organization constraints in definition 4.6;
4 **if** *Fusion contract with* A_i *is established successfully* **then**
5 **foreach** *input concept* $L_{i,j} \in \mathcal{L}_i$ *from agent* A_i **do**
6 in agent A_i call `TopDownConfiguration` ($L_{i,j}$);
7 **end**
8 in agent A_i call `BottomUpConfiguration` (R_i);
9 **end**
10 **end**

FIGURE 50 BOTTOM-UP CONFIGURATION ALGORITHM [78]

This algorithm is suitable for situations where it is desirable to organize fusion systems in response to unusual observations. It implements a simple self-organization rule: when an observation of a leaf variable is available, it searches the agents containing the corresponding service variable of this observed leaf node. For each of the matched agents, it starts to look for other agents who connect to it by its service variable and satisfy restriction 3.

In summary, DPNs as a distributed information fusion system has three limitations discussed in previous context and the following characteristics [97] :

- Reasoning for a single hypothesis variable is processed in a distributed way. The reasoning result reflects the entire available observations and is identical to the result from a single united Bayesian network.
- It does not need to check initial states consistency before the distributed system can work coordinately. Only local models need to be compiled in an independent way prior to run time.
- The information fusion process can work in an asynchronous way, which will improve the speed of reasoning process.
- The information fusion process is automatic and no pre-compilation or on-line check of the formulated structure to guarantee globally consistent inference.

As discussed before, for globally consistent inference in a distributed Bayesian network, there are three prerequisites:

- All of the sub Bayesian networks are organized into a tree structure.
- The tree structure satisfy running intersection property.
- Each node shared by two or more sub Bayesian networks should be a d-sep node.

DPNs satisfies those three conditions implicitly during the formulation of the network due to its strict requirements of individual sub Bayesian network stucture and adding new node to the system. More detailed discussion of DPNs can be found in [97].

4.6.4 Prior/Likelihood Decomposable Models

Prior/likelihood decomposable models (PLDMs) was proposed to infer sensor states and bias from noisy measurable data in large-scale complex distributed sensor networks in [80]. The author pointed out that this method can handle dynamic agent systems, such as adding/deleting agents and several damage situations, e.g., damaged communication links between two agents, bad data caused by failed sensors in a robust way. In sensor networks, each sensor is an agent. It divides all of the variables in the whole network into two types: ***observable variables*** and ***latent variables***. The observable variables are called measurable variables; each measurable variable corresponds to one of the sensors on one of the nodes. The latent variables are actually hidden variables which are called environment variables. The latent random variables characterize the state of the sensor networks' environment, such as the true temperature, the true pressure, the bias of a sensor itself, etc. All of the measurable variables are children of environment variables and the model needs to specify each measurable variable state probability conditioned on its corresponding hidden variables.

The basic idea of PLDMs is to give each node a subset of local priors. Those subsets of local priors are organized into a junction tree structure called external junction tree structure. In order to increase the robustness of node missing or communication link damages, prior for one node can be distributed to several different nodes as redundancy. The prior of one node is lost only when all of the nodes which include this node's prior are not available. The global prior distribution is obtained through message passing in the external junction tree similar to the normal junction tree belief propagation described in

Chapter III. Message passed between two agents is represented as a Prior/Likelihood (PL) factor of the shared variables.

A ***Prior/Likelihood (PL) factor*** for a set of environment variables C is a pair (π_c, λ_c) where

- π_c is a prior distribution for C: $prob(C)$.
- λ_c is a likelihood function: $prob(m|C)$, where m are the observation variables in one node.

In summary, PLDMs has the following limitations for applications other than distributed sensor networks.

- One sensor corresponds to one agent.
- All of the measurable variables are localized.
- All of the variables for the interfaces are belong to the measured variables.
- It doubles quantities of message passing among agents.
- It could break the rules of keeping privacy of each individual agent.
- In order to make globally consistent inferences, pre-compilation or on-line formulation and check of the formulated structure are needed

Detailed discussion of PLDMs can be found in [80].

4.6.5 Multiple Sectioned Bayesian Networks

The concept of multiple sectioned Bayesian networks was first proposed by Xiang et al. [98]. In this paper, Xiang gave the basic idea of MSBNs and its corresponding constraints and implementations. Following this paper, Xiang had carried out further researches on this topic and published a series of papers [96, 98-105] to refine and improve MSBNs

method. The following introduction to MSBNs is mostly based on Xiang's series of publications on this topic.

4.6.5.1 MSBNs Introduction

The basic idea of MSBNs is that it decomposes a big knowledge-based system into several sub networks. Each sub network represents its knowledge by a Bayesian network. By organizing those distributed sub Bayesian networks into a certain structure, globally consistent and efficient inferences are achievable in a distributed way without revealing all of the information of the system to each individual agent.

MSBNs is a derivative of Hyper tree MSDAG with further restrictions. The rigorous mathematical definition of MSBNs is as follows.

MSBNs: a MSBNs M is a triplet $M = (N, G, P)$. $N = \bigcup_{i=1,\cdots n} N_i$ is the total universe. Each $N_i \subset N$ is a set of variables. $G = \bigcup_{i=1,\cdots n} G_i$ is a Hyper tree MSDAG with a hyperlink set $S = \{S_i, \cdots S_m\}$. Each $G_i \in \{G_i, \cdots G_n\}$ is a Bayesian network with $dom(G_i) = N_i$ and a joint probability $P_{G_i}(N_i)$. $P = \prod_{i=1,\cdots n} P_{G_i}(N_i) \Big/ \prod_{i=1,\cdots m} P_{S_i}(S_i)$ is the joint probability distribution over the total universe N. Furthermore, for $\forall$ G_i and G_j are neighbors in G, the marginalizations of $P_{G_i}(N_i)$ and $P_{G_j}(N_j)$ on to their d-sep set intersection are identical. Each triplet $C_i = (N_i, G_i, P_i)$ is called a sub-net of M. C_i and C_j are neighbors if G_i and G_j are neighbors.

As mentioned before, there are three common prerequisites for consistent global inferences in distributed Bayesian networks: tree structure among sub Bayesian networks, d-sep set between neighbored sub Bayesian networks and running intersection property

of the tree structure. DPNs satisfies these three conditions implicitly during the establishment of the problem. For PLDMs and MSBNs, these three conditions are not embedded in the problem construction process and need to be checked manually.

4.6.5.2 Process of MSBNs Establishment

We get the general idea about MSBNs from the above content, now we need to know how to get a MSBNs for an actual system and how to use the MSBNs to make efficient and coherent inferences. In general, there are 5 steps to prepare a MSBNs of an actual system for distributed probabilistic inference:

- Get cause-effect relationship for a real system by extracting information from experts or experiments in different domains and form a set of sub Bayesian networks. Each sub Bayesian network corresponds to an intelligent agent.
- Check graph consistency of neighbored sub graphs through coordination among agents.
- Moralize the distributed Bayesian networks through coordination among agents.
- Triangulate the distributed Bayesian networks through coordination among agents.
- Establish Linked Junction Forest (LJF) for the distributed Bayesian networks.

Here a notional example is given to illustrate the 5 steps in detail.

Step 1: Get cause-effect relationships for a real system by extracting information from experts or experiments in different domains and form a set of sub Bayesian networks. Each sub Bayesian network is represented as an intelligent agent

Obtaining cause-effect relationships quantitatively in a real system is a very challenging job. It involves many experts and large set of data abstraction. Information from experts are subjective, thus whenever possible, it is better to get the quantitative cause-effect relationships from collected data set [106]. Fortunately, for MSBNs, one sub graph only

contains a small part of the whole system and normally in a specific domain; the whole task is decomposed into smaller subtasks in specific domains and each node in a sub Bayesian only involves a few connected nodes (Markov Blanket). Experts from specific domain can focus on their domain without the knowledge of other domains by establishing the causal-effect relationships and providing common interface to other sub domains.

Step 2: Check consistencies for two neighbored graphs

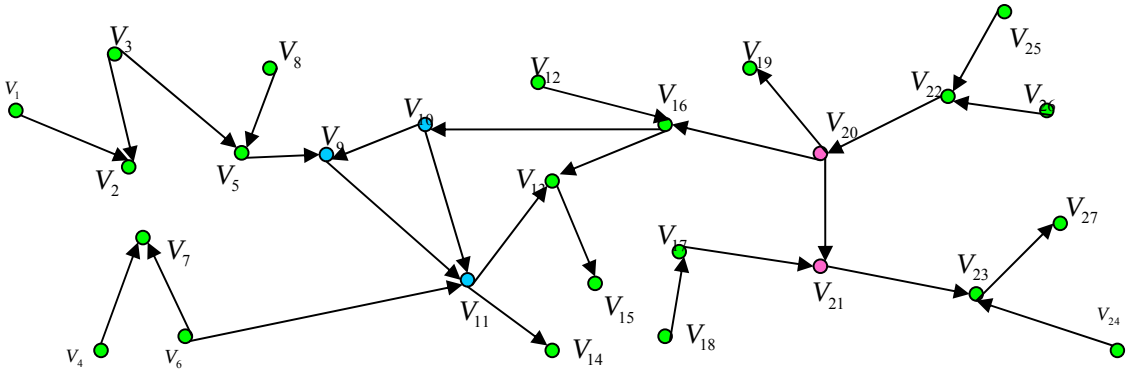


FIGURE 51 A NOTIONAL EXAMPLE FOR MSBNS

If the distributed Bayesian network is formed by sectioning a monolithic network, this step can be skipped. If individual sub graph is established by different experts independently, graph consistency by neighbored graphs needs to be checked through communication between agents. For example, when two agents detect conflict cause-effect relationships on their shared variables through communication, one of them will compromise to adopt the other's structure. Now we assume that we have an existed Bayesian network as shown in Figure 51. This whole Bayesian network is sectioned into three sub graphs in the way as shown in Figure 52. The resulted three sub graphs from the sectioning are shown in Figure 53. Since the three sub graphs are obtained by sectioning a monolithic Bayesian network, they are guaranteed to be graph consistent.

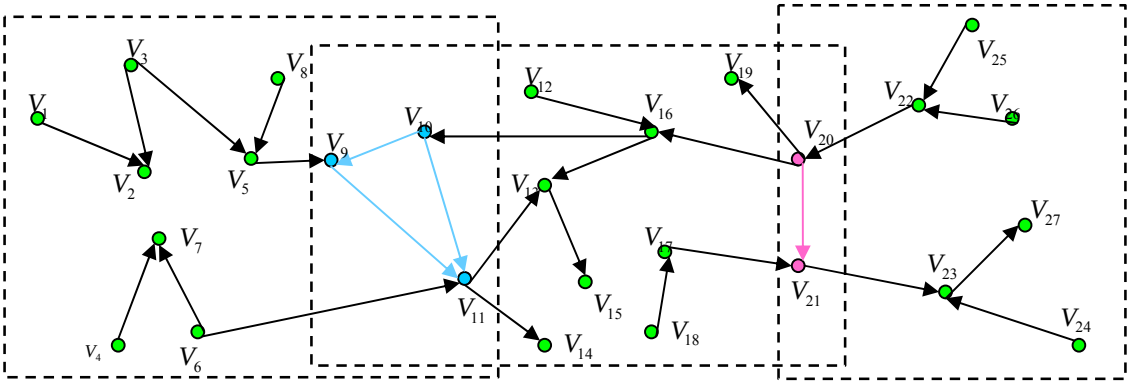


FIGURE 52 SECTION THE NOTIONAL EXMAPLE

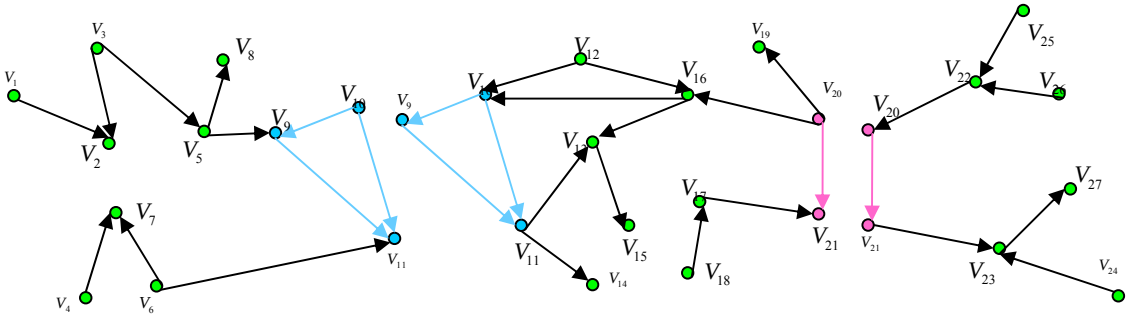


FIGURE 53 RESULTED THREE SUB GRAPHS FROM SECTIONING THE NOTIONAL EXMAPLE

Step 3: Moralize the distributed Bayesian networks through coordination among agents

Moralize each sub graph independently as described in Chapter 2. For two neighbored graphs, make sub graphs containing the intersection domain consistently by unifying the two sub graphs. For example, sub graphs G_1 and G_2 has an intersection set with the domain containing three nodes V_1, V_2 and V_3. After independent moralization, in graph G_1, V_1 is connected to V_2 and V_3 while in graph G_2, V_2 is connected to V_1 and V_3. After communicating between these two sub graphs, G_1 has a connection between two

common nodes V_1 and V_3 , which G_2 does not have; and then G_2 will add this connection. Similarly, G_2 has a connection between two common nodes V_2 and V_3 which G_1 does not have, and then G_1 will add this connection. Now, the resulted moralization sub graphs with the intersection domain in both of these two graphs are identical: V_1, V_2 and V_3 are connected pair wise.

The three sub graphs after cooperative moralization of the notional example is shown in Figure 54.

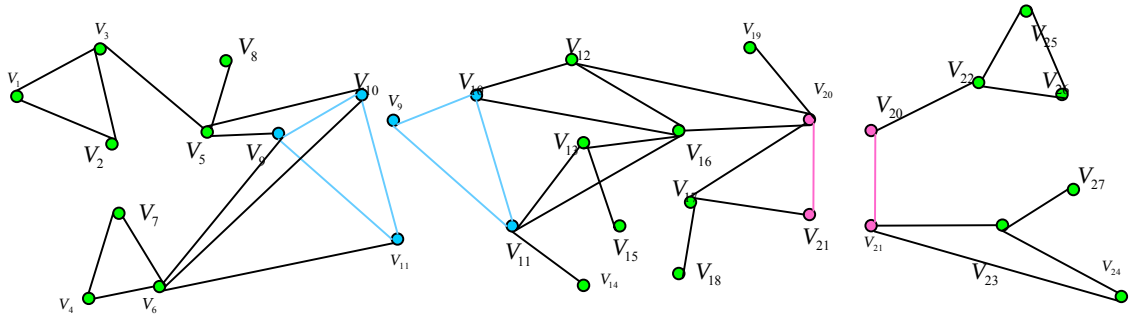


FIGURE 54 THREE SUBGRAPHS AFTER COOPERATIVE MORALIZATION OF THE NOTIONAL EXAMPLE

Step 4: Triangulate the distributed Bayesian networks through coordination among agents

The method used here is a distributed triangulation among different sub graphs without revealing privacy information of each sub graph. The input of this process is a set of undirected consistent moralized graphs and the output of this process is a corresponding set of chordal sub graphs which collectively defines a chordal super graph of the union of this set of sub graphs. By being chordal, a graph has at least one perfect elimination sequence.

Xiang gave recursive algorithms for each sub graph to do the cooperative triangulation for a hyper tree [102]. The entity initiating the algorithm can be a sub Bayesian graph agent or an operator controlling the system. If the initiator/caller is a sub graph agent, we denote it by A_c with corresponding sub graph G_c over domain N_c. The sub Bayesian agent called by the caller is denoted by A_0 with the corresponding sub graph G_0 over domain N_0. If A_0 has adjacent agents other than A_c, we denote them by $A_1, \cdots, A_n$ with the corresponding sub graphs $G_1, \cdots, G_n$ over domains $N_1, \cdots, N_n$. The separator between A_0 and A_c is denoted by $S_c = N_0 \cap N_c$. The separator between A_0 and A_i is denoted by $S_i = N_0 \cap N_i$, where $i = 1, \cdots, n$. Three algorithms are used recursively to achieve the coordinated triangulation of the sub cordal graphs: Depth First Eliminate algorithm, Distribute Dlink algorithm and CoTriangulate algorithm. Before introducing these three algorithms, we need to define a concept of restriction first.

Given a set F of links over a set N of nodes, a subset $E \subseteq F$ is a restriction of F to $M \subseteq N$ if $E = \{(V_i, V_j) \mid \forall V_i \in M, V_j \in M, (V_i, V_j) \in F\}$.

This definition is like to extract the complete connectivity information for a subset from a large node set.

Depth First Eliminate Algorithm

Set LINK= ϕ ;
If caller is an agent A_c, do
{
Receive a set F_c of fill-ins over S_c from A_c ;
Add F_c to G_0 ;
}
For each agent A_i ($i = 1, \cdots, n$), do
{
Eliminate N_0 in the order $(N_0 \setminus S_i, S_i)$ and denote the resultant fill-ins by F ;

Add F to G_0 and LINK;
Send A_i the restriction of F to S_i;
Call A_i to run Depth First Eliminate Algorithm and receive fill-ins F' over S_i from A_i when finished;
Add F' to G_0 and LINK;
}
If caller is an agent A_c, do
{
Eliminate N_0 in the order $(N_0 \setminus S_c, S_c)$ and denote the resultant fill-ins by $F_c^{'}$;
Add $F_c^{'}$ to G_0 and LINK;
Send A_c the restriction of LINK to S_c;
}

Distribute Dlink Algorithm

If caller is an agent A_c, do
{
Receive a set F_c of fill-ins over S_c from A_c;
Add F_c to G_0;
}
Set LINK to the set of all fill-ins added to G_0 so far;
For each agent A_i ($i = 1, \cdots, n$), do
{
Send A_i the restriction of LINK to S_i
}

CoTriangulate Algorithm

Choose an agent A_* arbitrarily
Call A_* to run Depth First Eliminate algorithm
After A_* has finished, call A_* to run Distribute Dlink algorithm

---

By using these algorithms, the resulted three sub graphs are shown in Figure 55. For this example, the result after cooperative triangulation is the same as the result from the cooperative moralization. In general, it is not the case. However, even for this special

case, it does not mean this step is useless. By doing this process, it proves that the resulted graphs satisfy the following two requirements [96]:

- Requirement 1: For each hyper node in the hyper tree of MSBNs, the triangulated moral graph G over N for the hyper node must be in a structure such that for every hyperlink L_i incident to the hyper node G_i, G_i is eliminable in the order $(N_i \setminus L_i, L_i)$.
- Requirement 2: For each pair of adjacent hyper nodes in the hyper tree of MSBNs, the corresponding d-sep set must be connected identically in the two triangulated moral graphs. That is, the two triangulated graphs must be graph-consistent.

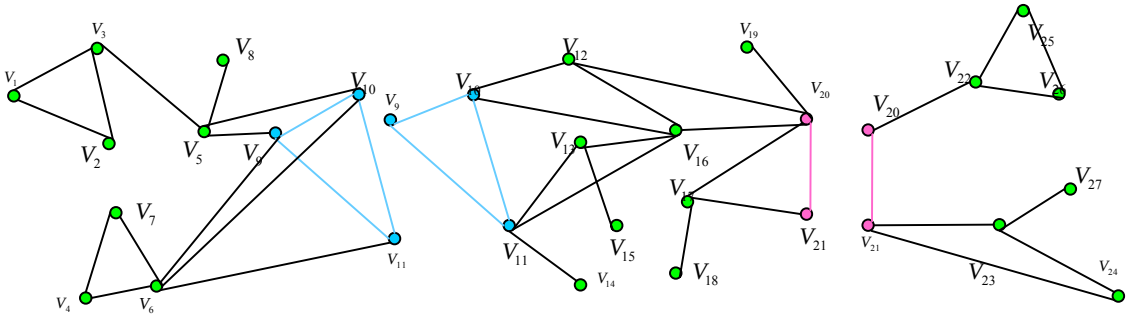


FIGURE 55 THE THREE SUB GRAPHS AFTER COOPERATIVE TRIANGULATION OF THE NOTIONAL EXAMPLE

Step 5: Establish Linked Junction Forest (LJF) for the distributed Bayesian networks

After cooperative triangulation, the sub graphs are ready to be used to establish junction trees for efficient inference reasoning. In this step, we establish a junction tree for each individual sub graph. The processes of establish junction trees for the three sub Bayesian networks in the notional example is shown in Figure 58, Figure 56 and Figure 57. We also establish a junction tree for each d-sep set between two sub graphs as well to prepare for efficient reasoning. After this step, a linked junction forest is formed for the whole

system. After the Linked Junction Forest is established, the MSBNs is ready for on-line inferences according to local observations at every sub graph without revealing each sub graph's internal structure or parameters. This inference process will execute 4 operations to establish a global and consistent inferences for any hypothesis contained in any sub graph's cliques [103].

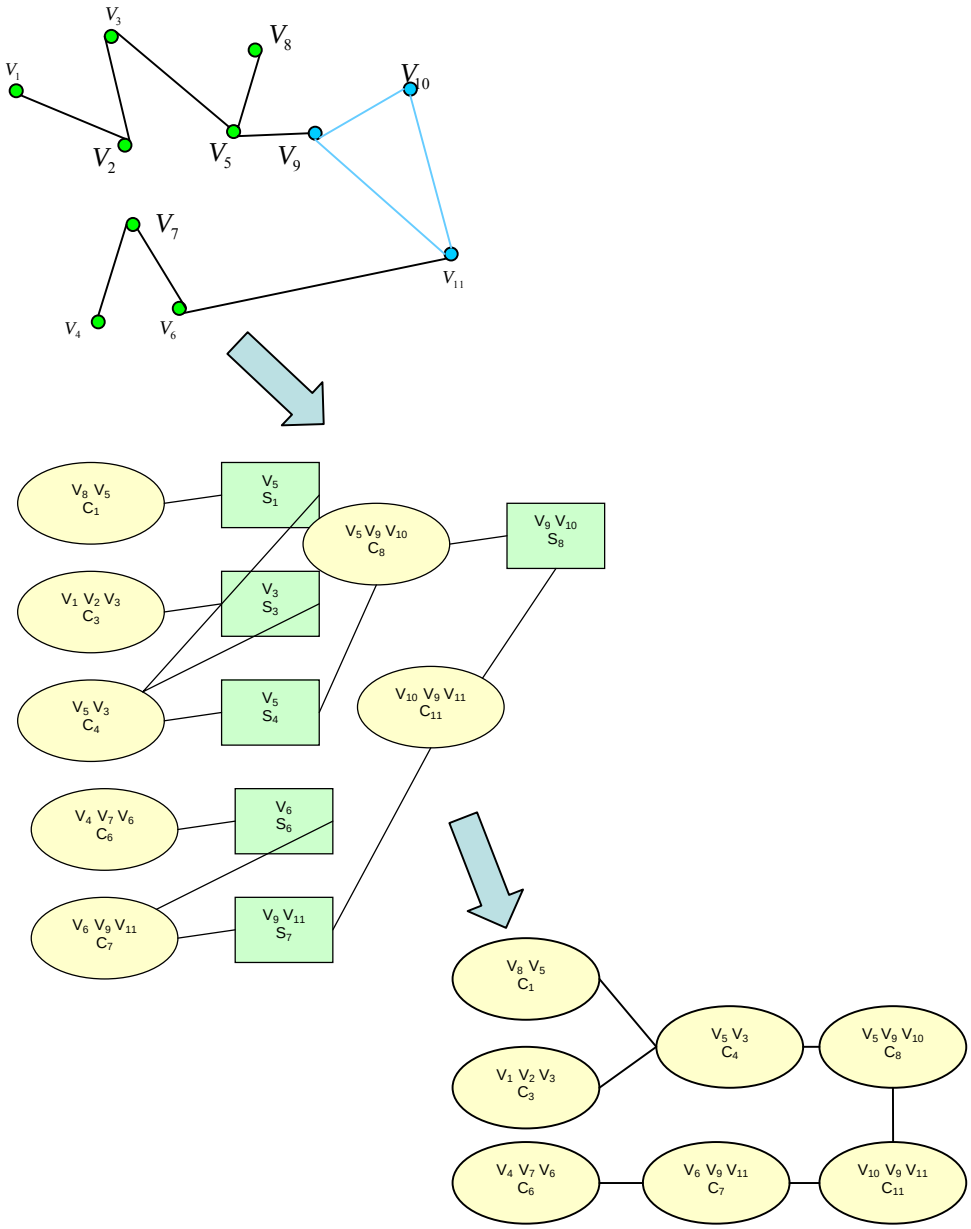


FIGURE 56 JUNCTION TREE FORMULATION OF SUB GRAPH 1

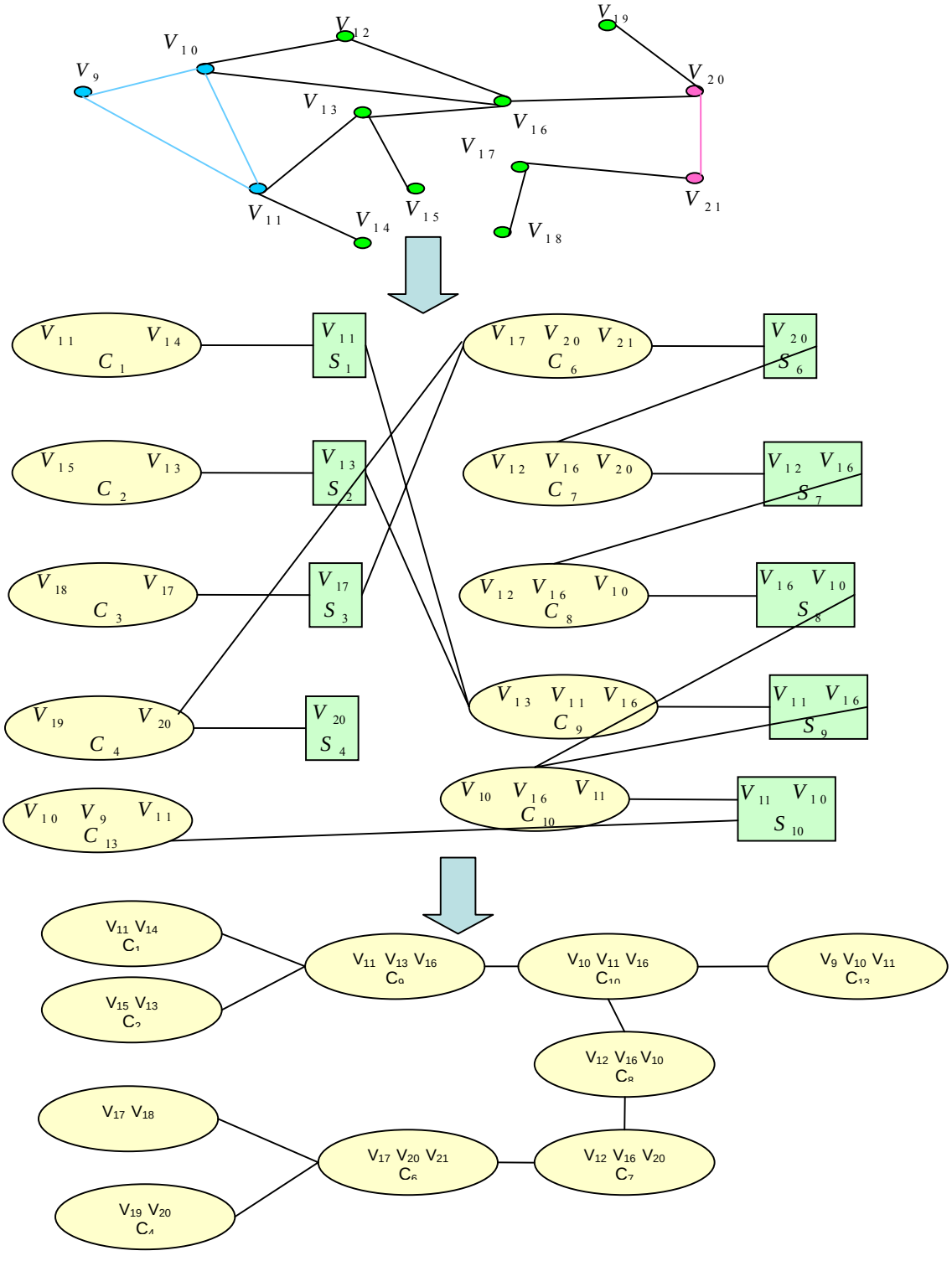


FIGURE 57 JUNCTION TREE FORMULATION OF SUB GRAPH

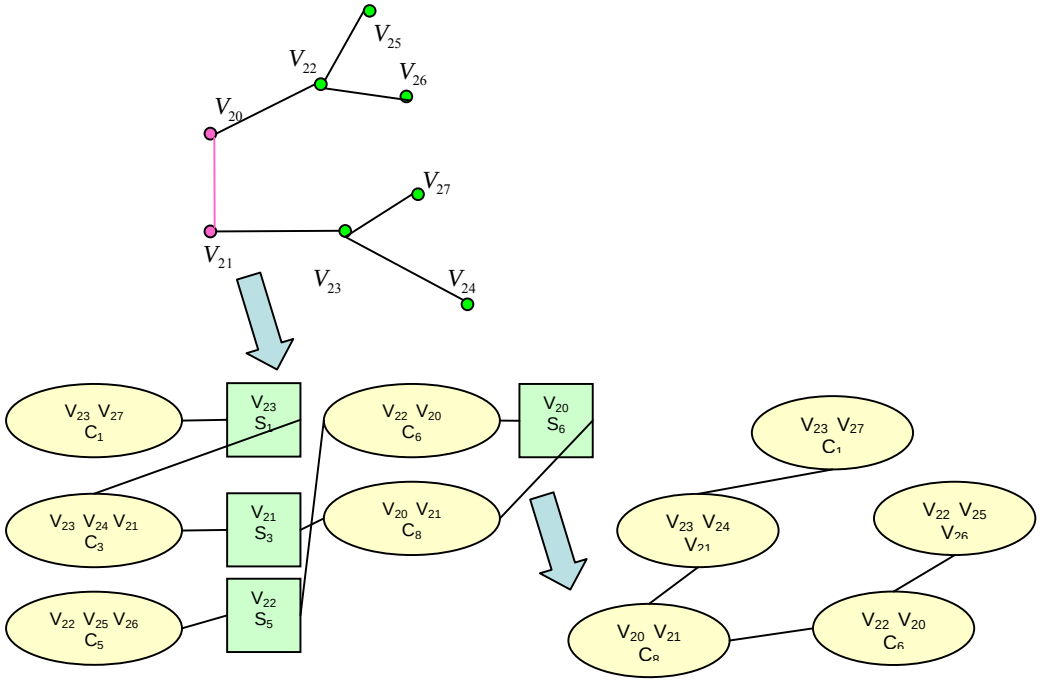


FIGURE 58 JUNCTION TREE FORMULATION OF SUB GRAPH 3

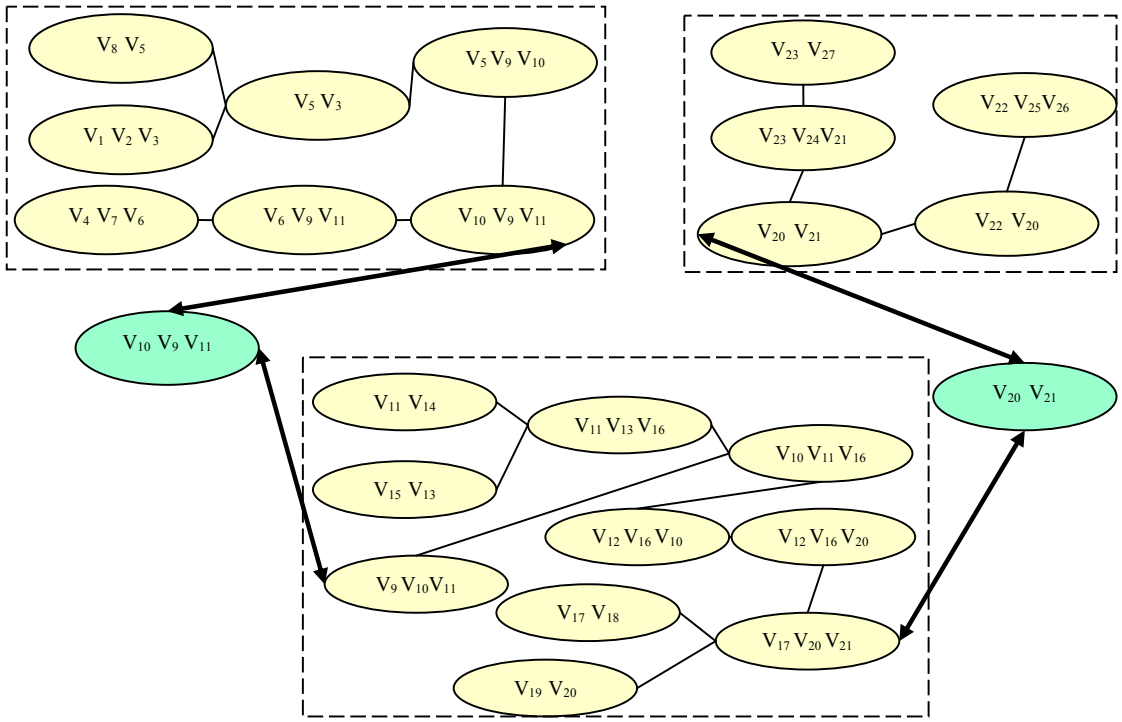


FIGURE 59 THE LINKED JUNCTION FOREST OF THE NOTIONAL EXAMPLE

Operation 1: AbsorbThroughLinkage

Let l be a linkage in a linkage tree L between two joint trees T_i and T_j corresponding to two neighbored sub graphs G_i and G_j with domains N_i and N_j respectively. Let C_i and C_j be two chosen cliques in T_i and T_j respectively satisfying $l \subseteq C_i$ and $l \subseteq C_j$. Let $B_l^*(l)$ be the extended linkage belief associated with l, and $B_{C_i}^*(l)$ be the extended linkage belief on l projected from C_j.

When **AbsorbThroughLinkage** is called to execute on C_i to absorb from C_j through linkage l, perform the following two steps:

- Updating the chosen clique C_i belief by $B_{C_i}^{'}(C_i) = B_{C_i}(C_i) \times B_{C_i}^*(l) / B_l^*(l)$.
- Updating linkage belief by $B_l^{*'}(l) = B_{C_i}^*(l)$.

Operation 2: UnifyBelief

Let T be a Junction Tree for a sub graph and C be a cluster in T. When UnifyBelief is called to execute, initiate a full propagation from C (definition of full propagation is defined in Chapter II) as described in Chapter II for a junction tree of an individual Bayesian network.

Operation 3: UpdateBelief

Let T_i and T_j be two adjacent junction trees corresponding to two neighbored sub graphs G_i and G_j with domains N_i and N_j respectively. And let L be the linkage tree between T_i and T_j. When UpdateBelief is called to execute on T_i relative to T_j, two steps need to follow:

- For each linkage l in L, call a clique C_i in T_i satisfying $l \subseteq C_i$ to perform AbsorbThroughLinkage.
- Then Perform UnifyBelief at T_i.

Operation 4: CollectBelief

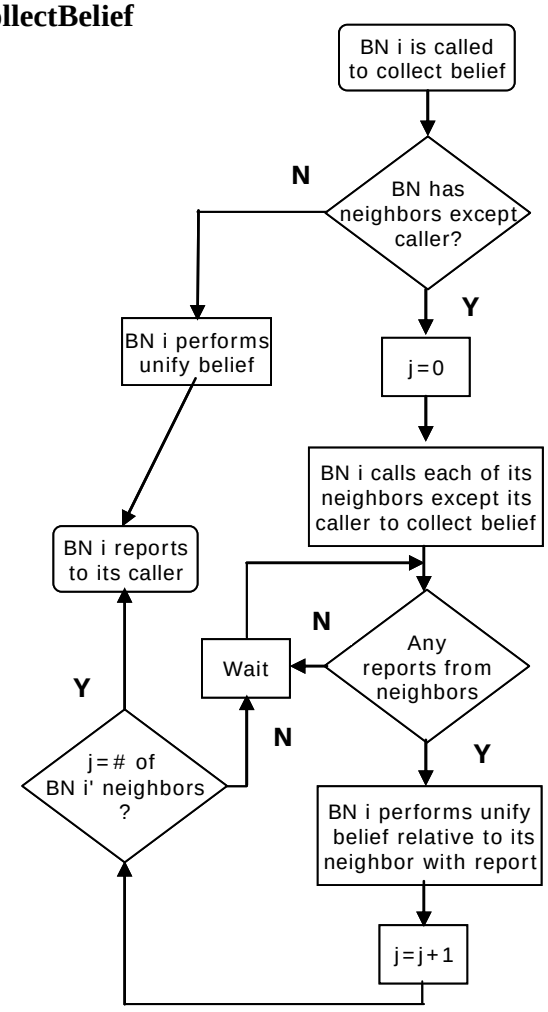


FIGURE 60 BAYESIAN NETWORK I PERFORMS COLLECTBELIEF PROCESS

Let T_i be a junction tree in a linked junction forest. When the T_i gets a CollectBelief call from the system or one of its adjacent junction trees T_j, T_i performs the two steps below:

- If T_i has no other adjacent junction trees except T_j, it performs UnifyBelief and return.
- Otherwise, for each other adjacent junction tree T_k except T_j, call CollectBelief in T_k. After T_k finishes, T_i performs UpdateBelief relative to T_k.

The flow chart of a Bayesian network performing CollectBelief process is shown in Figure 60.

Operation 5: DistributeBelief

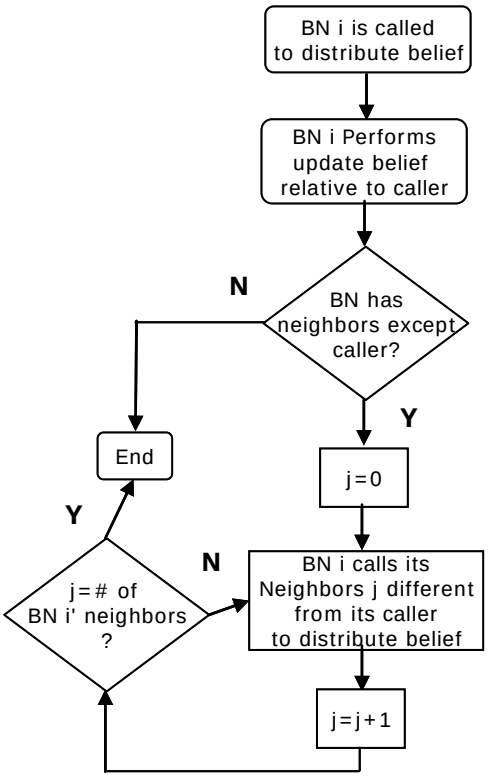


FIGURE 61 BAYESIAN NETWORK I PERFORMS DISTRIBUTEBELIEF PROCESS

Let T_i be a junction tree in a linked junction forest. When T_i gets a DistributeBelief call from the system or one of its adjacent junction trees T_j, T_i performs the two steps below.:

- If the call from one of its adjacent junction trees T_j , it performs UpdateBelief and relative to T_j .
- For each other adjacent junction tree T_k except T_j , call DistributeBelief in T_k .

The flow chart of a Bayesian network performing DistributeBelief process is shown in Figure 61.

Operation 6: CommunicateBelief

When the system calls one of junction tree T_i in a linked junction forest, CollectBelief is executed at T_i, followed by a DistributedBelief execution at T_i.

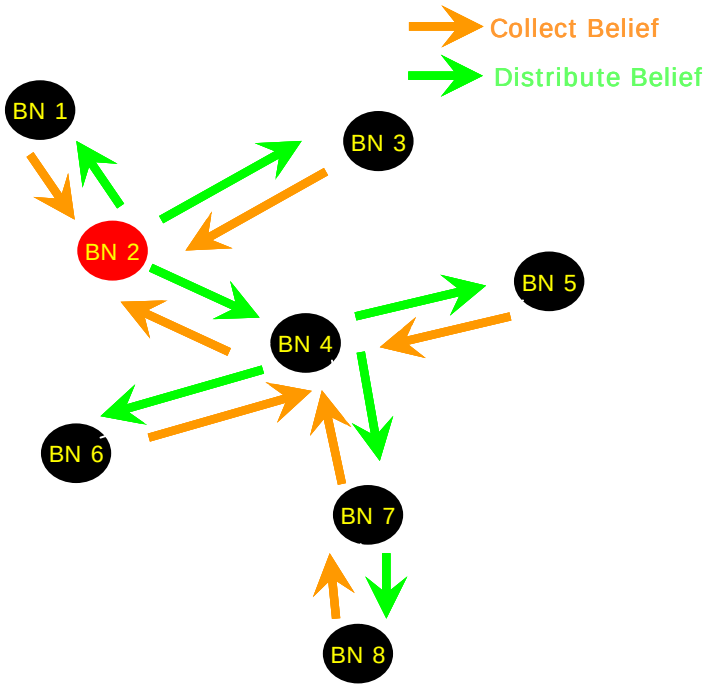


FIGURE 62 COMMUNICATEBELIEF MESSAGE PASSINGS IN A MSBNS

CommunicateBelief is similar to the idea of a full propagation in an individual Bayesian network. The difference is that CommunicateBelief transfers messages among junction trees while a full propagation in an individual Bayesian network transfers messages among cliques. The whole CommunicateBelief message passing among a MSBNs is

shown in Figure 62. Total number of message passing is equal to $2(n-1)$, where n is the number of agents in the whole system.

Xiang proved that a linked junction forest is globally consistent after the operation of CommunicateBelief in this linked junction forest [103]. Each junction tree in the system is ready to make its local inferences based on the global available information.

In summary, MSBNs has the following characteristics and limitations:

- Model general complex systems in a distributed way.
- Support general structure for any individual sub-network.
- Support general interface with multiple common variables.
- Support partial asynchronous message passing.
- Allow correct calculation of posterior probability distributions for any variables at one time.
- Keep partial privacy of individual sub-networks.
- Complicated to learn and implement.
- In order to make globally consistent inferences, pre-compilation or on-line formulation and check of the formulated structure are needed.

Detailed discussion of MSBNs can be found in [96, 98-105].

4.6.6 Summary of Distributed Bayesian Networks

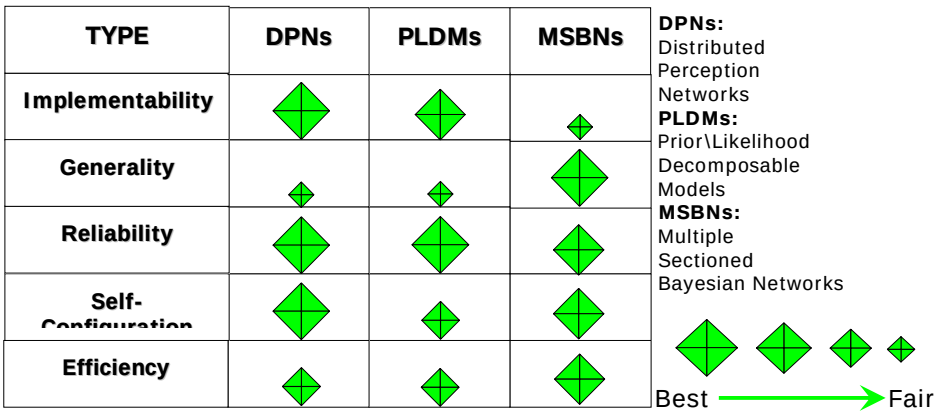
In the previous context, three types of distributed Bayesian networks are introduced. Each of them has its own advantages and disadvantages. Comparing them quantitatively is a time-consuming task which is beyond the scope of this research. Here, the comparisons focus on five attributes in a qualitative way: implementability, generality, reliability, self-configuration, and efficiency.

DPNs is relatively simple to be implemented. The connections among different agents can be established at run time (self-configuration). It does not need prior compilation and communications are established only when necessary, thus it has high efficiency for one query variable at one time of a network with simple structure. However, DPNs has three strict requirements for its structures as listed before, therefore, it is only suitable for some special applications.

PLDMs originates from sensor networks where one sensor corresponds to one agent and all of the observations are localized. It improves its reliability through double message passing among agents and redundant hidden variables prior distributions. It does not support self-configuration, therefore, pre structure compilation is needed.

MSBNs is the most difficult approach to be implemented. However, it supports more general complex systems. Its reliability and self-configuration capability can be improved through careful agent structure designs, which will be discussed later in this chapter. Communication efficiency of MSBNs can also be improved through breaking shared variables between sub Bayesian networks into smaller linkages. In addition, individual sub Bayesian network agent can keep privacies from other sub Bayesian networks and a set of multiple queries can be made simultaneously in MSBNs.

TABLE 3 COMPARISONS OF DISTRIBUTED BAYESIAN NETWORKS



As a summary, the comparison results are shown in Table 3. From this table, we can see DPNs is superior to PLDMs and MSBNs in every aspect except generality and efficiency due to its strict requirements for adding new node and one query variable at one time. PLDMs has high reliability due to redundant prior distributions. Both DPNs and PLDMs are not suitable for general large-scale complex systems. MSBNs is suitable for general large-scale complex systems. Both of PLDMs and MSBNs require pre-compilation or on-line self-organization and structure checking.

4.6.7 Hypothesis 2.2 and Hypothesis 2.3

From the comparisons above, we can see for a general large-scale complex system, MSBNs is the best choice of distributed inference engine if on-line self-organization and structure checking can be implemented in an automatic way. Two corresponding hypotheses are following:

- ***H2.2: If Multiple Sectioned Dynamic Bayesian Networks (MSDBNs) can be established for a large-scale complex system,***
  - ***each sub-Bayesian network can make its own decisions for its local system relatively independently;***
  - ***globally consistent inferences can be made by each sub Bayesian network through limited message passing among sub Bayesian network agents;***
  - ***sub-Bayesian network agents can self-organize into MSDBNs structures and make inference automatically in a distributed way when a system is partially damaged.***
- ***H2.3: MSDBNs can be embedded into the hybrid control architecture as an internal distributed inference engine to handle system uncertainties.***

4.6.8 Online Self-Organization and Structure Checking of MSDBNs

As discussed before, MSDBNs as a distributed Bayesian network, its structure needs to satisfy three conditions:

- All of the sub Bayesian networks are organized into a tree structure.
- The tree structure satisfy running intersection property.
- Each node shared by two or more sub Bayesian networks should be a d-sep node.

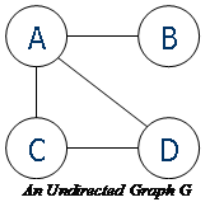
In the following sections, detailed discussions about how a distributed Bayesian network organizes itself into an MSDBNs structure which fulfills the above three conditions in an automatic and distributed way.

4.6.8.1 Automatic Spanning Tree Topology Formation

The idea behind the spanning tree topology formation for MSDBNs is to connect different sub Bayesian networks to form a loop-free topology, i.e. a tree in which every sub Bayesian network can reach every other sub Bayesian network directly or indirectly.

Connectivity of all sub Bayesian networks can be expressed in a connectivity matrix $C(G)$. $C(G)$ involves a number of rows and cells equivalent to the number of sub Bayesian networks in the whole system. Each cell representing a direct connection between two nodes receives a value 1; otherwise, it receives a value 0. For an undirected graph, connection in the network is bi-directional and $C(G)$ is symmetric. The reachability matrix $R(G)$ records whether or not at least there is one path (of any length) between sub Bayesian network i and sub Bayesian network j. An entry $r_{ij} = 1$ in $R(G)$ if i is reachable from j. A spanning tree is an acyclic tree structure; every node can reach every another node while every node can not reach itself as a cycle. The structure of $R(G)$ for a spanning tree is a symmetric matrix in which all of the entries are 1 except

the entries on the diagonal which are zero. An undirected graph G and its corresponding connectivity matrix $C(G)$ and reachability matrix $R(G)$ are shown in Figure 63 as an example.



$c(i,j)$	A	B	C	D
A	0	1	1	1
B	1	0	0	0
C	1	0	0	1
D	1	0	1	0

$C(G)$

$r(i,j)$	A	B	C	D
A	1	1	1	1
B	1	0	1	1
C	1	1	1	1
D	1	1	1	1

$R(G)$

FIGURE 63 A GENERAL UNDIRECTED GRAPH AND ITS CORRESPONDING CONNECTIVITY MATRIX AND REACHABILITY MATRIX

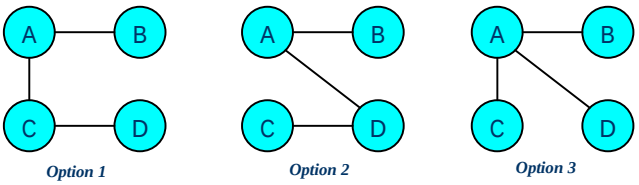


FIGURE 64 SPANNING TREES OF THE GENERAL UNDIRECTED GRAPH SHOWN IN FIGURE 63

There are one/multiple spanning trees for a specific connected graph. For example, the general undirected graph in Figure 63 has three spanning threes as shown in Figure 64. How to choose the best spanning tree for a specific connected graph is an optimization problem. The most common used spanning tree optimization for a connected undirected graph is to choose the simplest spanning tree with minimum summation of edge weights.

Assume that an undirected graph $G = (V, E)$ consists of a set V of vertices and a set $E \in V \times V$ of edges with unordered pairs of vertices. Assign each edge of the connected graph a weight $w(e_i)$, $e_i \in E$. The objective of optimization of spanning tree for this graph is to get a tree $T(V', E')$ for graph G which minimizes

$$W(T) = \sum_{\forall e_i \in E'} w(e_i),\ where\ V' = V\ and\ E' \subseteq E \tag{4.9}$$

For example, let a general undirected graph G with weighted edges as shown inFigure 65. G has three different spanning trees with different summations of edge weights as shown in Figure 66. Apparently, option 3 is the one with minimum summation of edge weights.

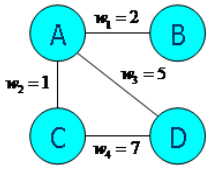


FIGURE 65 A GENERAL UNDIRECTED GRAPH WITH WEIGHTED EDGES

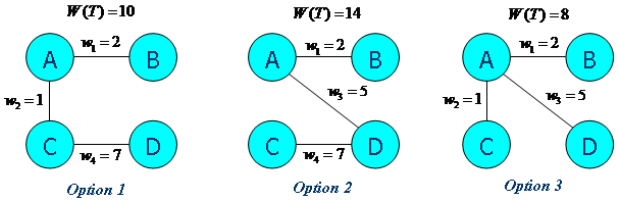


FIGURE 66 THREE OPTIONS OF SPANNING TREES WITH DIFFERENT SUMMATIONS OF EDGE WEIGHTS

Listing all possible spanning trees for an undirected graph G to find the one with minimum summation of edge weights is the most computational consuming way. There are many more efficient algorithms for spanning tree optimization, such as Kruskal's algorithm, Prim's algorithm, Boruvka's algorithm, Hybrid algorithms[107], etc. Those algorithms are based on the following lemma.

Lemma: Let $G = (V, E)$, and V_1, $V_2 \subseteq V$, $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \varnothing$. If e is the smallest edge connecting V_1 and V_2, then e is part of the minimum spanning tree.

Kruskal's Algorithm

```
sort the edges of G in increasing order by length
keep a sub graph S of G , initially empty
for each edge e in sorted order
    if the end points of e are disconnected in S
    add e to S
return S
```

Prim's Algorithm

```
 Initially, let T be a single vertex v
  while (T has fewer than n vertices)
  {
     find the smallest edge connecting T to G \ T
     add it to T
}
```

Boruvka's Algorithm

```
make a list L of n trees, each a single vertex
   while ( L has more than one tree)
   {
      for each T in L , find the smallest edge connecting T to G \ T
      add all those edges to the MST
      (causing pairs of trees in L to merge)
}
```

Those three algorithms can be combined together to get some hybrid algorithms which may perform better than one single algorithm.

Distributed Spanning Tree Optimization Algorithm

The above three algorithms are easy to understand and implement. However, all of them are centralized algorithms and are not suitable for large-scale distributed systems. Therefore, a distributed spanning tree formation algorithm is needed. A distributed algorithm for minimum-weight spanning tree was proposed by Gallarger, al. [108] for an

undirected graph with distinguished weighted edges. In this algorithm, only message passing between neighbors is involved until the minimum-weight spanning tree is constructed. For an undirected graph with finite nodes N and edges E, the number of messages passed between neighbors is at most $5N\log_2^N + 2E$, and the time is $\mathrm{O}(N\log N)$. Actions among different nodes are partially asynchronous, and the whole process can be initiated at any node or any multiple nodes.

For an undirected graph with N nodes and E edges, initially, as the Boruvka's Algorithm, each node is a fragment. Each fragment has its own level and a unique identity. For agents in JADE, the start name of each fragment can be the agent name and its level starts from 0. Each fragment finds its minimum-weight outgoing edge asynchronously with other fragments.

Each node has a state SN $\{Sleep, Find, Found\}$ and each edge has a state SE $\{Branch, Rejected, Basic\}$.

- *Sleep* : initial state of a node at the start of the algorithm.
- *Find* : a node participating in a fragment's search for the minimum-weight outgoing edge.
- *Found* : a node belongs to a fragment which has already found its minimum-weight outgoing edge.
- *Branch* : the edge is already confirmed to be in the minimum-weight spanning tree.
- Re *jected* : the edge is already confirmed not to be in the minimum-weight spanning tree.
- *Basic* : the edge is neither confirmed to be in the minimum-weight spanning tree nor confirmed not to be in the minimum-weight spanning tree.

Originally, each fragment starts with the state *Sleep* and *Level* 0. Each edge starts with the state *Basic* . For a node tries to find its minimum outgoing edge, it sends out a *test* message. A node receives a *test* message and sends a *reject* or *accept* message back to the sender node.

- *test* message: test if this edge is an outgoing edge.
- *accept* message: confirm that the edge is an outgoing edge.
- *reject* message: confirm that the edge is not an outgoing edge.

Initially, all edges connecting to a node form the basic edge set for this node and all nodes choose the minimum weight edge from their basic edge set and send a message called *connect* message with the information of the edge identity (identities of two end nodes connecting this edge) and the fragment's name and level. A *connect* message is expressed as $connect(FI, FL, MWEI(EI1, EI2))$, where FI indicates fragment identity; FL indicates fragment level and MWEI indicate minimum weight edge identity which includes end node identity 1 EI1 and end node identity 2 EI2. At the same time, the node state is changed into state *Found* and waiting for a response from the fragment at the other end of the selected edge.

Combination Rules of Two Fragments

A fragment F_1 *with* (FI_1, FL_1) *accepts a message* $Connect(FI_2, FL_2, MWEI_2(EI1_2, EI2_2))$ *from another fragment* F_2.

if $FL_1 < FL_2$

Fragment F_2 *immediately absorbs* F_1 ;

The new combined fragment F_{new} *'s level is* FL_2 ;

else if $FL_1 = FL_2$

F_1 *starts to find its* $MWEI_1(EI1_1, EI2_1)$;

if $MWEI_1(EI1_1, EI2_1) = MWEI_2(EI1_2, EI2_2)$

F_1 *and* F_2 *are combined into a new fragment* F_{new} *with level* $FL_2 + 1$*;*

else

F_1 *waits until* $FL_1 < FL_2$ *or*

$\left(\left(FL_1 < FL_2\right) \text{ and } \left(MWEI_1(EI1_1, EI2_1) = MWEI_2(EI1_2, EI2_2)\right)\right)$

end

end

The waiting combination rule reduces communications by only allowing smaller or equal level fragments to join other bigger or equal level fragments, because communication required for a fragment to find its minimum weight outgoing edge is proportional to the fragment size. That the waiting will not cause any deadlock of this algorithm was justified in [108] .

It is easy for a fragment with one node to find a minimum outgoing edge. However, for a fragment with several nodes, it is a bit more complicated. All of the nodes in the fragment need to cooperate with each other. When two fragments are combined with each other into a new fragment through an edge, starting from these two end nodes of the edge, a message with the new fragment identity, new fragment level and *find* command will be spread outwards to the whole fragment. A node in the fragment receives this message starts to find its minimum weight outgoing edge from its basic edge set. In the basic edge set, if a node gets a message from the other side of an edge with the same fragment name, this edge in both of the end nodes will be assigned to the state *rejected* . After this step, every node in one fragment finds its own minimum weight outgoing edge. The last step is to find the minimum weight outgoing edge for the whole fragment. It works this way: each leaf node sends its minimum weight outgoing edge and its identity to its neighbors; the inward edge compares its own minimum weight edge with all of the received minimum weight outgoing edges and chooses the smallest one and sends the smallest one with its node identity to its own inward node. One node as a center point or end point will get the smallest outgoing edge for the whole fragment, and then it will traces back where

the smallest edge comes from. The node containing the smallest weighted outgoing edge will send out the *connect* message to the fragment which contains the other end node of this edge. A relative simple example will show this whole process clearly.

Assume a fully connected undirected graph *G* as shown in Figure 67 with its edge weight matrix as shown in Figure 68.

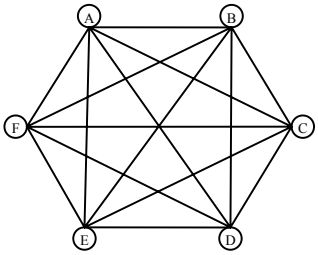


FIGURE 67 A FULLY CONNECTED UNDIRECTED GRAPH G

$w(i,j)$	A	B	C	D	E	F
A	∞	0.6813	0.3046	0.1509	0.4966	0.3420
B	0.6813	∞	0.1897	0.6979	0.8998	0.2897
C	0.3046	0.1897	∞	0.3784	0.8216	0.3412
D	0.1509	0.6979	0.3784	∞	0.6449	0.5341
E	0.4966	0.8998	0.8216	0.6449	∞	0.7271
F	0.3420	0.2897	0.3412	0.5341	0.7271	∞

FIGURE 68 EDGE WEIGHT MATRIX OF GRAPH G

At the start point, A, B, C, D, E and F as individual fragments find their own minimum weight outgoing edges AD (0.1509), BC (0.1897), CB (0.1897), DA (0.1509), ED (0.4966), and FB (0.2897) respectively.

A sends a *connect* message to D. D finds that the level of fragment of A is equal to its own fragment level and that fragment A has the same minimum weight outgoing edge; it increases its fragment level by 1 and changes its fragment identity to DA; it marks edge

DA as *branch*; it changes its state to *find* and initiates *find* action; it sends an *find* message with its new fragment level and fragment identity back to A. A accepts this message and changes its own fragment level to 1 and fragment identity to DA; it marks edge AD as *branch*; it changes its state to *find* and starts to execute *find* action.

B sends a *connect* message to C. C finds the level of fragment of B is equal to its own fragment level and fragment B has the same minimum weight outgoing edge; it increases its fragment level by 1 and changes its fragment identity to CB; it marks edge CB as *branch*; it changes its state to *find* and initiates *find* action; it sends an *find* message with its new fragment level and fragment identity back to B. B accepts this message and changes its own fragment level to 1 and fragment identity to CB; it marks edge BC as *branch*; it changes its state to *find* and starts to execute *find* action. After that, the connection is shown in Figure 69.

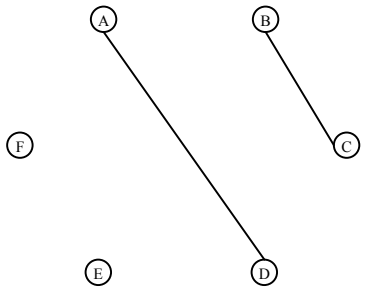


FIGURE 69 CONNECTIONS AFTER A COMBINES D AND B COMBINES C

E sends a *connect* message to D. D finds the level of fragment of E is less than its own fragment level, then absorbs node E; it marks edge DE as *branch*; it sends a *find* message with its fragment level and fragment identity to E. E accept this message and changes its fragment level and fragment identity; it changes its state to *find*; it marks edge ED as *branch*; it starts to execute *find* action.

F sends a *connect* message to B. B finds the level of fragment of F is less than its own fragment level and it absorbs node E; it marks edge BF as *branch* ; it sends a *find* message with its fragment level and fragment identity to F. F accept this message and changes its fragment level and fragment identity; it changes its state to *find* ; it marks edge FB as *branch* ; it starts to execute *find* action. After this step, the connections are shown in Figure 70.

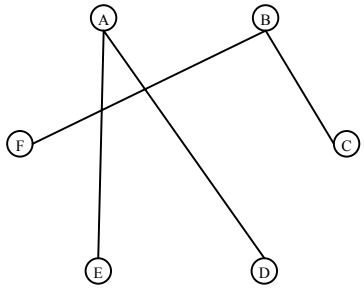


FIGURE 70 CONNECTIONS AFTER AD ABSORBS E AND BC ABSORBS F

Now, A, B, C, D, E and F are trying to find their own minimum weight outgoing edges independently.

A finds its minimum weight outgoing edge AC (0.3046) in the basic edge set and sends a *test* message to C. C accepts this message and sends an *accept* message back to A. A accepts this message and confirms that edge AC (0.3046) is its own minimum weight outgoing edge and changes its state to *found* .

B finds its minimum weight outgoing edge BA (0.6813) in the basic edge set and sends a *test* message to A. A accepts this message and sends an *accept* message back to B. B accepts this message and confirms that edge BA (0.6813) is its own minimum weight outgoing edge and changes its state to *found* .

C finds its minimum weight outgoing edge CA (0.6813) in the basic edge set and sends a *test* message to A. A accepts this message and sends an *accept* message back to C. C accepts this message and confirms that edge CA (0.6813) is its own minimum weight outgoing edge and changes its state to *found* .

D finds its minimum weight outgoing edge DC (0.3784) in the basic edge set and sends a *test* message to C. C accepts this message and sends an *accept* message back to D. D accepts this message and confirms that edge DC (0.3784) is its own minimum weight outgoing edge and changes its state to *found* .

E finds its minimum weight outgoing edge EA (0.4966) in the basic edge set and sends a *test* message to A. A rejects this message due to the same fragment identity, marks edge AE as *rejected* and sends a *reject* message back to E. E accepts this message and marks edge EA as *rejected* . E finds it second minimum weight outgoing edge EF (0.7271) and sends a *test* message to F. F accepts this message and sends an *accept* message back to E. E accepts this message and confirms that edge EF (0.7271) is its own minimum weight outgoing edge and changes its state to *found* .

F finds its minimum weight outgoing edge FC (0.3412) in the basic edge set and sends a *test* message to C. C rejects this message due to the same fragment identity, marks edge CF as *rejected* and sends a *reject* message back to F. F accepts this message and marks edge FC as *rejected* . F finds it second minimum weight outgoing edge FA (0.3420) and sends a *test* message to A. A accepts this message and sends an *accept* message back to F. F accepts this message and confirms that edge FA (0.3420) is its own minimum weight outgoing edge and changes its state to *found* .

Now, A, B, C, D, E and F find their own minimum weight outgoing edges respectively. A, D and E need to cooperate to find a minimum outgoing edge for fragment DA while B, C and F need to cooperate to find a minimum outgoing edge for fragment CB.

A and E send *report* messages to D respectively. D compares A's minimum weight outgoing edge, E's minimum weight outgoing edge and its own minimum weight outgoing edge; it finds AC is the minimum from these three edges; it informs A about that. A accepts this message and marks edge AC as *branch*; it sends a *connect* message to C.

F and C send *report* messages to B respectively. B compares F's minimum weight outgoing edge, C's minimum weight outgoing edge and its own minimum weight outgoing edge; it finds CA is the minimum from these three edges; it informs C about that. C accepts this message and marks edge CA as *branch*; it sends a *connect* message to A.

C accepts the *connect* message from A; it finds they have the same fragment level and the same minimum weight outgoing fragment edge; it increases its fragment level by one and changes its fragment identity to CA; it changes its state to *find*; it starts to execute *find* action; it sends a *find* message to A and B with the new fragment level and identity. A accept the *find* message from C and changes its state to *find*; it starts to execute *find* action and sends a *find* message to D. D accept the *find* message from A and changes its state to *find*; it starts to execute *find* action and sends a *find* message to E. E accepts this message, changes its state to *find* and starts to execute *find* action. C accepts a *find* message from B, changes its state to *find* and starts to execute *find* action.

Now A, B, C, D, E and F are trying to find their own minimum weight outgoing edges again as described before. In this step, since A, B, C, D, E and F are already connected as a tree; no one will find an outgoing edge. However, they need to cooperate with each other to get the information that the algorithm is over. E sends a *report* message to D; D sends a *report* message to A; F sends a *report* message to B; B sends a *report* message to C; C sends a *report* message to A. Now A knows the algorithm is over and it sends a *over* message outwards of the tree to let every node knows the algorithm is over. The final spanning tree of graph G is shown in Figure 71.

This undirected graph includes $N = 6$ nodes which are fully connected, therefore, there are $E = \frac{N \times (N-1)}{2} = \frac{6 \times (6-1)}{2} = 15$ edges. At most, there are $5N \log_2^N + 2E = 5 \times 6 \times \log_2^6 + 2 \times 15 \approx 108$ messages involved in order to get the minimum weighted spanning tree. The actual number of messages involved in the above process for this problem is 64 < 108.

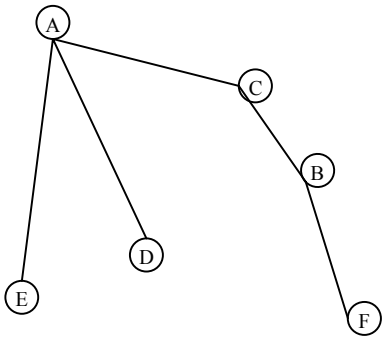


FIGURE 71 THE FINAL SPANNING TREE OF GRAPH G

Based on the distributed minimum-weight spanning tree algorithm described above, Awerbuch gave a new algorithm with an $O(V)$ running time in 1987 [109]. Awerbuch's

algorithm has a slightly increased number of total message passings but with the same level of communication complexity $O(E + V \log V)$. Also originated from Gallager's algorithm, in 1993, Garay et al. proposed a algorithm with further improvment of running time complexity to $O(Diam + n^{0.614})$, where $Diam$ is the maximum number of edges for connecting any two nodes in the connected graph [110]. Garay et al. pointed out that this algorithm could increase the number of total meassage passings implicitly without further discussion of communication complexity.

For multiple sectioned Bayesian networks, each sub Bayesian network can be taken as a node in the above distributed spanning tree optimization algorithm. It does not reveal detail information of each sub Bayesian network, as long as a connection between these two sub Bayesian networks and the corresponding cost/weight are known. Communication quality between two sub Bayesian networks can be used as one factor of determining edge weight between them.

4.6.8.2 Distributed D-Sep Set Satisfaction

As we know, after moralization, all of the parents of a node need to be connected. D-sep set says that at least one graph needs to contain all parents of any individual shared variable. It is the same as Markov property in an undirected graph, which is: set A and set B are conditionally independent by given set C if and only if no path can connect elements in A and elements in B without through elements in C. It is called global Markov property in contrast to local Markov property for Markov Blanket addressed in Chapter III. Global Markov property provides the basis of scarce communication between different sub Bayesian networks, i.e., it is sufficient and complete that only the shared variables' information is transferred between different sub Bayesian networks.

Xiang proposed a method to verify d-sep set condition of each shared node by multiple sub Bayesian networks in a cooperative way through message passing between

neighbored sub Bayesian networks. Those transferred messages reveal only partial information on the adjacency of a shared node in an local sub Bayesian network [111]. Xiang showed that the total time complexity for the cooperative verification of d-sep set for all shared variables is $O\left(n^2\left(k^3\ s^2 + k\ s\ t\right)\right)$, where n is the total number of sub Bayesian networks; k is the maximum number of nodes in a sub Bayesian network interface; s is the maximum number of agents adjacent to any given agent on the spanning tree and t is the maximum cardinality of a node adjacency in a local sub Bayesian network.

However, for an automatic process, after the verification of d-sep node process, if some shared nodes are verified as non d-sep nodes, the whole system can not make globally consistent inferences. How to pick up non d-sep nodes and change them into d-sep nodes automatically is a more challenging task. Currently, to the best knowledge of the author through literature search, there does not exist an efficient way to perform d-sep node formation automatically. The following method is based on the modification of Xiang's cooperative d-sep set verification method [111] and it takes one more step further to process d-sep node formation.

Proposition: Let a public node V in a spanning tree DAG union of G be a d-sep node, then no more than one local DAG of G contains private parent nodes of$V$ [111].

For one shared variable, this method includes three aspects:

- At most one sub Bayesian network contains all its private parents.
- If only one sub Bayesian network contains its private parents, then this sub Bayesian network also has to contain all its public parents.
- If it does not have private parents, at least one sub Bayesian network contains all its public parents.

The minimum required information about a shared node's parents in a specific sub Bayesian network is composed of four pieces of information $parInfo = \{V, BN, s, \Omega\}$: this shared node's name V, the sub Bayesian network name BN, a logistic value s indicating if this sub Bayesian network contains its private parents or not, its public parent set Ω in this sub Bayesian network. For addition of two nodes' parent information, there are three rules as described as follows:

Combination of Two Nodes' Parent Information

Let $parInfo_1 = \{V_1, BN_1, s_1, \Omega_1\}$ *and* $parInfo_2 = \{V_2, BN_2, s_2, \Omega_2\}$.

if $V_1 \neq V_2$

$parInfo_{12} = \{\{V_1, BN_1, s_1, \Omega_1\}, \{V_2, BN_2, s_2, \Omega_2\}\}$

else if $s_1 = 1$

$parInfo_{12} = \{V_1, BN_1, s_1, \Omega_1 \cup \Omega_2\}$

else if $s_2 = 1$

$parInfo_{12} = \{V_2, BN_2, s_2, \Omega_1 \cup \Omega_2\}$

else if $\Omega_1 \supseteq \Omega_2$

$parInfo_{12} = \{V_1, BN_1, s_1, \Omega_1\}$

else if $\Omega_1 \subset \Omega_2$

$parInfo_{12} = \{V_2, BN_2, s_2, \Omega_2\}$

else if $cardinality(\Omega_1) \geq cardinality(\Omega_2)$

$parInfo_{12} = \{V_1, BN_1, s_1, \Omega_1 \cup \Omega_2\}$

else

$parInfo_{12} = \{V_2, BN_2, s_2, \Omega_1 \cup \Omega_2\}$

end

return $parInfo_{12}$

Combination of a Set of Nodes' Parent Information and One Node's Parent Information

Let $parInfoSet = \{parInfo_1, \cdots, parInfo_m\}$ *and* $parInfo$.

Initialize $parInfoSet\_new = parInfoSet$;

Initialize flag=false

for i=1 to m
if $parInfo_i.V = parInfo.V$
flag=true;
$parInfo = parInfo_i + parInfo$ *;*
Delete $parInfo_i$ *from* $parInfoSet\_new$ *;*
Add parInfo to $parInfoSet\_new$ *;*
break
end
end
if flag=false
Add parInfo to $parInfoSet\_new$
end
return $parInfoSet\_new$

Combination of Two Sets of Nodes' Parent Information

Let $parInfoSet_1 = \{parInfo_{11}, \cdots, parInfo_{1m}\}$ *and* $parInfoSet_2 = \{parInfo_{21}, \cdots, parInfo_{2n}\}$ *,*
then
initialize $parInfoSet_{12} = parInfoSet_1$
for i=1 to n
$parInfoSet_{12} = parInfoSet_{12} + parInfo_{2i}$
end

Initially, each sub Bayesian network stores a set of all of its public nodes' parent information according to this specific sub Bayesian network structure.

A sub Bayesian network is chosen as the root point arbitrarily and it collects public nodes' parent information from its leaf sub Bayesian networks. The leaf sub Bayesian networks send their own set of public nodes' parent information to their corresponding neighbors. A sub Bayesian network combines its own public nodes' parent information set and the received public nodes' parent information sets from all of its downstream neighbors and sends this combined set to its upstream neighbor. After the root sub Bayesian network gets all messages from its neighbors, it has all public nodes' required parent information in order to form d-sep set condition. And it starts to distribute the

collected information and force each sub Bayesian network to form d-sep node for each shared node by sending the collected information and "FormD-SepSet" message to its downstream neighbors. When a sub Bayesian network gets a "FormD-SepSet" message, it starts to execute "FormD-SepSet" action. As we know, the collected information includes every shared node's required public parent information and the sub Bayesian network's name which contains all of its parents.

FormD-SepSet Algorithm

A sub Bayesian network BN_i with a set of shared variable Ω_i accepts a FormD-SepSet message containing an information set $parInfoSet = \{parInfo_1, \cdots, parInfo_n\}$

for each $V_i \in \Omega_i$
  flag=false;
  if there exists $parInfo_j.V = V_i$ in parInfoSet
    flag=true;
  end
  if flag
    if $parInfo_j.BN \neq BN_i$
      Make all of V_i's private parents in BN_i public;
    else
      Make BN_i includes all public nodes $parInfo_j.\Omega$;
      Delete $parInfo_j$ from parInfoSet;
    end
  else
    Make all of V_i's private parents in BN_i public;
  end
end

Send a FormD-SepSet message containing parInfoSet to its downstream neighbor sub Bayesian networks.

However, after this execution, some private nodes are forced to be public; the whole system needs to redo the whole d-sep set process for those new joined public variables. The whole process stops until no new private parents are forced to be public. This process may take long time to converge. In reality, we may skip this step and use non d-sep nodes.

Inferences in the whole system with non d-sep nodes shared by two sub Bayesian networks are not globally consistent, but may close enough to global consistency.

After the optimal spanning tree with d-sep set is formed, next step is to force running intersection property in the spanning tree.

4.6.8.3 Distributed Running Intersection Satisfaction

As defined in Chapter III, running intersection property says a shared variable by two sub Bayesian networks in a tree structure must be contained in every sub Bayesian network on the route connecting those two sub Bayesian networks. Running intersection property provides information consistency by only tranfering messages between neighbored networks. Paskin and Guestrin formulated a distributed algorithm for running intersection satisfaction, which evolves all variable information in the whole network [80].

Paskin and Guestrin's Algorithm

For each connection between two neighbored sub Bayesian networks $G_i\left(N_i, E_i\right)$ and $G_j\left(N_j, E_j\right)$, define the variables reachable to G_j from G_i by

$$R_{ij} \triangleq N_i \cup \bigcup_{k \in n(i): k \neq j} R_{ki} \tag{4.10}$$

where $n(i)$ is the total number of G_i's neighbors in the tree structure. G_i computes R_{ij} by collecting the variables that can be reached through each neighbored sub Bayesian network but sub Bayesian network G_j and adding its own local variable N_i; then it sends R_{ij} as a message to G_j. If a sub Bayesian network receives two reachable variable messages that include some variables set X, then it knows that it must also contain X. Formally, a sub Bayesian network G_i calculates its domain iteratively as:

$$N_{i_{new}} \triangleq N_{i_{old}} \cup \bigcup_{k,l \in n(i): k \neq l \neq i,} \left(R_{ki} \cap R_{li} \right) \tag{4.11}$$

Apparently, Paskin and Guestrin's Algorithm involves all variables information and the messages transferred between two sub Bayesian networks are too heavy. Inspired from Paskin and Guestrin's algorithm, a distributed running intersection property satisfaction algorithm for a tree structure which only involves a small sub set of public variables is proposed in the following section.

An Efficient Distributed Running Intersection Property Satisfaction Algorithm

Assume a sub Bayesian network G_i with domain $N_i = N_{ip} \cup N_{is}$, where N_{ip} represents private variable set; N_{is} represents public variable set and $N_{ip} \cap N_{is} = \Phi$. $N_{is} = N_{ij} \cup \overline{N}_{ij}$, where N_{ij} represents the shared public variables by sub Bayesian network G_i and G_j.

For each connection between two neighbored sub Bayesian networks $G_i\left(N_i, E_i\right)$ and $G_j\left(N_j, E_j\right)$, define the variables reachable to G_j from G_i indirectly by

$$R_{ij} \triangleq \overline{N}_{ij} \cup \bigcup_{k \in n(i): k \neq j} R_{ki} \tag{4.12}$$

where $n(i)$ is the total number of G_i's neighbors in the tree structure. G_i computes R_{ij} by collecting the variables that can be reached indirectly through each neighbored sub Bayesian network, but sub Bayesian network G_j and adding its own local public and non-shared with G_j variables $\overline{N}_{ij}$; then it sends R_{ij} as a message to G_j. If a sub Bayesian network receives two reachable variable messages that include some variable set X, then it knows that it must also contain X. Formally, a sub Bayesian network G_i calculates its domain iteratively as:

$$N_{i_{new}} \triangleq N_{i_{old}} \cup \bigcup_{k,l \in n(i): k \neq l \neq i,} \left(R_{ki} \cap R_{li} \right) \tag{4.13}$$

In this algorithm, only the information of a small set of public variables which are not shared by two neighbored sub Bayesian networks is transferred between two sub Bayesian network agents and it reduces communication complexity. Formulate the whole process of running intersection property satisfaction, three operations are involved.

Operation 1: CollectIndirectlyReachableAndPublicVars

Let BN_i represent a sub Bayesian network in a tree structure. When BN_i gets a CollectIndirectlyReachableAndPublicVars call from the system or one of its neighbor sub Bayesian network BN_j, BN_i performs the two steps below:

- If BN_i has no other neighbor sub Bayesian network except BN_j, it sends its R_{ij} to BN_j.
- Otherwise, for each other neighbor sub Bayesian network BN_k except BN_j, call CollectIndirectlyReachableAndPublicVars in BN_k. After BN_k finishes, BN_k calculates its domain N_k according to equation 4.13 and also sends R_{ki} to BN_i, where R_{ki} is calculated according to equation 4.12. When BN_i gets all messages from its neighbors except BN_j, it calculates its domain N_i according to equation 4.13 and also sends R_{ij} to BN_j, where R_{ij} is calculated according to equation 4.12.

Operation 2: DistributeIndirectlyReachableAndPublicVars

Let BN_i be sub Bayesian network in a tree structure. When BN_i gets a DistributeIndirectlyReachableAndPublicVars call with R_{ji} from sub Bayesian network BN_j, BN_i performs the two steps below:

- BN_i calculates its domain N_k according to equation 4.13.

- If BN_i has other neighbor sub Bayesian network BN_k except BN_j, it sends R_{ik} to BN_k and the DistributeIndirectlyReachableAndPublicVars message, where R_{ik} is calculated according to equation 4.12.

Operation 3: ForceRunningIntersectionProperty

When a sub Bayesian network BN_i in a tree structure is chosen arbitrarily by the system or is initiated by itself, CollectIndirectlyReachableAndPublicVars is executed at BN_i, followed by a DistributeIndirectlyReachableAndPublicVars execution at BN_i.

After the Operation ForceRunningIntersectionProperty is over, the whole tree structure satisfies the running intersection property.

4.6.8.4 Summary of Online Self-Organization and Structure Checking of MSDBNs

In the above sections, the detailed discussions about distributed spanning tree optimization, distributed d-sep set formation and distributed running intersection satisfaction are given. All of them are automatic, partially asynchronous and only involving public information. Among the three processes, distributed d-sep set formation and distribution running intersection satisfaction are two iterative processes, which stop until no changes exist in both of them. The combination of those three processes and MSDBNs distributed belief updating algorithm forms an automatic distributed multiple sectioned dynamic Bayesian networks inference engine, which is suitable for a general large-scale complex system with dynamic structure changing and communication damages.

In the next chapter, a summary of the methodology and process proposed in the first four chapters will be presented.

CHAPTER V

SUMMARY OF PROPOSED METHEDOLOGY

In this chapter, a summary of the main proposed methodology and procedure is presented to help readers refresh their minds and get to the essences of this research quickly.

5.1 Proposed Methodology

The general research objective of this dissertation is to develop a comprehensive, generalized framework for the control system design of large-scale complex systems under significant uncertainties. It includes two interactive parts: distributed control architecture design, and distributed inference engine design. The designed inference engine will be embedded into the control architecture to provide state estimations for controllers.

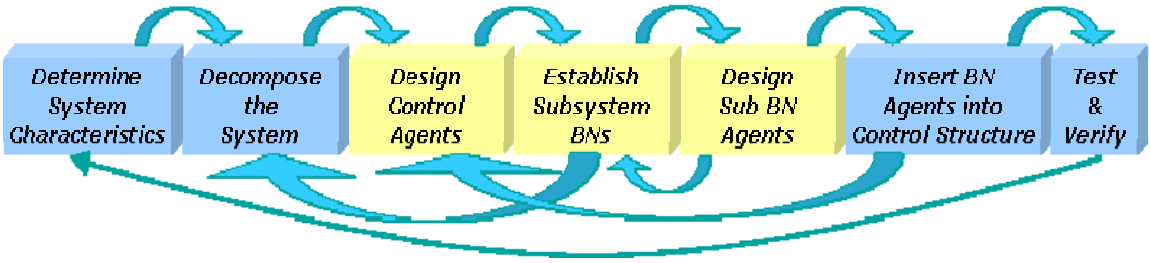


FIGURE 72 PROCESS OF CONTROL SYSTEM DESIGN FOR LARGE-SCALE COMPLEX SYSTEMS

A summary of the procedure of designing the control system for large-scale complex systems is given in Figure 72.

5.1.1.1 Step 1: Determine System Characteristics/Check System Constraints & Requirements

A large-scale complex system consists of a large set of sub systems/components. All of those sub systems/components may have different local objectives to achieve. However, for a meaningful and useful system, all of the sub systems/components interact with each other to achieve some common system level objectives. The purpose of designing controllers is to make the system move toward the direction of the desired results/goals in a faster, more efficient and more robust way. Hence, for the control system design of a large-scale complex system, most of the all, the designers need to have knowledge of the requirements of the system. More specifically, before designing the control system, the designers need to ask themselves a set of questions. What are the inputs for the system? What are the outputs for the system? What are the requirements for the outputs? How are the desired outputs maintained based on the inputs? Are there disturbances of the inputs and what are the characteristics of the disturbances? Are there constraints for the states/inputs of the system? Constraints may be regarded as boundaries that define the range of conditions within which the system may operate. Constraints may decide which control strategies are acceptable and how the controller can improve system performance. For example, for a process control system, constraints may arise from several sources, including quality considerations, physical limitations, equipment capacities, avoiding equipment overload, etc. In some systems, the functional goal is to maintain safety. Therefore, safety is part of the overall objective as well as potentially part of the constraints [112]. For an aircraft control system design, the constraints include structure bearing limits, aerodynamic instabilities, mechanical laws, high safety considerations, etc. There are several important requirements for the system, such as an effective fault tolerance in order to allow the unit to continue safe operation for several hours in the presence of faults, reducing the pilot's workload, and preventing the crew from inadvertently exceeding the aircraft's controllability limits [112].

This step is very important, and there are many methods to analyze system constraints and requirements. However, this step is not the focus of this research. In the application part of this dissertation, we will assume that the requirements and constraints are known.

5.1.1.2 Step 2: Decompose the system

The purpose of this step is to divide a large-scale complex system into smaller, more manageable pieces and address each piece in a relatively isolated manner. A system can be decomposed spatially by its components' spatial positions, or fuctionally by using function analysis, or by using both methods simultaneously, which depends on the structures of the physical system and the objectives of the control system. There are two processes to do control system decomposition: top-down approach and bottom-up approach[36].

For a top-down approach, a centralized system model is explicitly constructed firstly, and then it is decomposed into several subsystems by using structural properties presented in the system model. This method was implemented in [36, 113, 114].

For a bottom-up approach, there is no explicit centralized model. A virtual and conceptual model may be used as a reference. Subsystems are formed first and the designers check the relations between the subsystems to form higher level systems. This approach is adopted in [115-117].

These two processes can be combined in certain ways, such as establishing a rough centralized model according to the information from subsystems first, decomposing the rough centralized model into clearer and more convenient subsystems, and checking the relations between the subsystems to refine higher level systems.

5.1.1.3 Step 3: Establish Control Architecture & Control Agents

Step 3.1:Abstraction/Design Individual Agent

In this step, the aim is to design the internal logic and interface for each piece, which can satisfy the autonomy requirement as individual agent. Munroe and Luck claimed that three sets of motivations: ***domain motivations, constraint motivations, and social motivations*** are sufficient enablers of autonomy in three key areas of agent operation [37]:

- ***Domain Motivations*** represent the concerns and tasks that make up the agent's functional role in the system it belongs to.
- ***Constraint Motivations*** control the ways in which domain motivations are satisfied. It is similar to the concept of control constraints in a conventional way.
- ***Social Motivations*** determine the manner in which an agent interacts with other agents. The interface of each agent mainly depends on its relationships with other agents inhabited in the same system.

To be more specific, first of all, the designers need to know the objectives and structures of information it can obtain. Relationships between its inputs and its outputs satisfying its local objectives and constraints need to be established explicitly.

Step 3.2: Organize All Agents

The objective of this step is to define and manage the interrelationships among various portions of the system. This step is the most challenging part in the procedure. However, if the interactions between the agents are sparse, the difficulty of establishing the relationships among various sections is alleviated considerably. Fortunately, since a complex system frequently takes the form of hierarchy, a component in a subsystem will often interact directly with components in another subsystem in a sparse way. Therefore, generally there are intensive interactions between the components within the same subsystem while very sparse interactions between components in different subsystems. Furthermore, most of the communications between components in different subsystems

can be through a higher level agent instead of communicating with each other directly, which makes the establishment of relationships easier to organize. Each agent has its inputs and outputs from the first step (abstraction). The second step (organization) determines where the inputs come from and where the outputs go. Using an analogy to object-oriented programming (OOP), each agent can be taken as a class. The inputs and outputs for each agent can be listed by using conventional and understandable naming strategies, which helps find the interrelationships among the agents. Some guidelines for OOP are similar to this and can be adopted for the design of agent-based control systems (e.g., the guidelines provided by Eckel [38] for OOP).

Step 3.3: Add Auxiliary Agents

This step is not necessary if the first three steps produce properly matched interfaces. There are three types of matching to consider when designing an agent-based control system:

- The interfaces between the different control agents must be matched.
- The interfaces between the control agents and the physical model must be matched.
- The interfaces between the control system and some other systems must be matched.

For example, in some situations, interfaces between different interacting agents do not match accurately and some auxiliary agents need to be included to permit proper transfer of information between the entities.

Step 3.4: Create Replication Rings for Critical Agents

Choose critical agents according to specific applications and their corresponding decomposition schemes and create as many as necessary replications for each critical agent. Add more functions to those replications such as monitoring their neighbor

replications, communicating with other replications, forming new ring configurations among the replications, etc.

5.1.1.4 Step 4: Establish Sub System Bayesian Networks

A Bayesian network is a graphical model for cause-effect probabilistic relationships among a set of variables. For each sub system, the designers need to know how many variables are in the system and what they are. And the designers also need to investigate the cause-effect relationships among those variables. Usually, this is not an easy job and needs a lot of expertise in each specific area. Structure and parameter learning of Bayesian networks is needed if a large quantity of historical data is available. A too sparse structure in a Bayesian network cannot fully capture the system relationships and a too dense structure will dramatically slow down the inference efficiency. Biased prior information will deviate the reasoning results from actual values. For a large-scale complex system, the whole system is divided into many sub systems. Each individual sub Bayesian network is constructed by the sub domain experts who know how this sub system works. A Bayesian network is an acyclic graphical model, i.e., it can not handle directed cycles. After the Bayesian network is established, the acyclic property needs to be checked to guarantee consistent inference for this individual Bayesian network. For a time-evolving system, DBN is necessary to capture the system stochastic characteristics. In this dissertation, a two-time slice homogenous DBN with Boyen-Koller algorithm is implemented.

5.1.1.5 Step 5: Design Each Sub Bayesian Network as an Agent and Design Its Corresponding Internal Logics and Series of Behaviors for DSTO, DDSSS, DRIS and DBP

From the previous step, cause-effect relationships are established for each sub Bayesian network quantitatively. However, as discussed in Chapter IV, in order to make globally

consistent inferences for the whole system, the sub Bayesian networks have to be organized into a MSDBNs structure which satisfies three conditions:

- All of the sub Bayesian networks are organized into a tree structure.
- The tree structure satisfies the running intersection property.
- Each node shared by two or more sub Bayesian networks should be a d-sep node.

If the whole system structure is static and the communication system is robust, these sub Bayesian networks can be precompiled into a MSDBNs structure, which will be used in the following belief propagation. However, in a large-scale complex system, the system structure may change over time and the communication system could be damaged or blocked under certain conditions. In order to handle a system with dynamic structure and unstable communications, agents for sub Bayesian networks need to have the capability to be organized into a new MSDBNs structure online automatically. By establishing communication schemes and internal logics for each sub Bayesian network agent to implement the four algorithms: Distributed Spanning Tree Optimization (DSTO), Distributed D-Sep Set Satisfaction (DDSSS), Distributed Running Intersection Satisfaction (DRIS), and Distributed Belief Propagation (DBP) as discussed in detail in Chapter IV, automatic online self-organization of MSDBNs could be realized. The four algorithms will be implemented into serial behaviors of each sub Bayesian network agent, such as defining what type of messages an agent can receive, how to behave differently according to different messages, and how to initiate certain actions and communications with its neighbors, etc.

5.1.1.6 Step 6: Insert MSDBNs into the Control Architecture

The distributed Bayesian networks are used to do probabilistic state estimations and provide this information to the control system. The control system collects this information and uses it to control the system toward the desired direction. What and how

is the information among Bayesian network agents and control agents exchanged? Some intermediate agents may need to be established to make them work together consistently. Each sub Bayesian network is represented as an agent. The way that a Bayesian network agent communicates with other agents is the same as communications between control agents described in Chapter II.

5.1.1.7 Step 7: Test & Verify the Control System

After the first run of the design process, the designed control system needs to be tested and verified. If it does not satisfy some system requirements, the designers need to go back to apply modifications. Design of the control system is an iterative process.

After the whole process is done, the control system with the physical system is shown in Figure 73. Basically, the whole system consists of three relatively distinguishable parts: Bayesian network agents, control agents, and physical systems.

Bayesian network agents collect information from sensors and controllers, and then perform state estimations of components/subsystems and feed that information to the controllers. Bayesian network agents also communicate with each other to "use up" the available information to make the inferences as globally consistent as possible. However, each Bayesian network agent can make its own inference relatively independently by using its local available information. Since each Bayesian network is represented as an intelligent agent, therefore, through careful communication scheme and internal logic design described in step 5, smaller MSDBNs can be formed online automatically to make more accurate reasoning, as compared to independent individual Bayesian network when one or more sub Bayesian networks are not available to the whole system. Although certain sequences exist in actions for DSTO, DDSSS, DRIS and DBP, there are many chances for parallel computing. Logical replication ring organizations provide critical agent fault tolerance and improve the robustness of the whole system. Controller

agents use the component state estimations from the inference engine and make decisions to reconfigure the whole system.

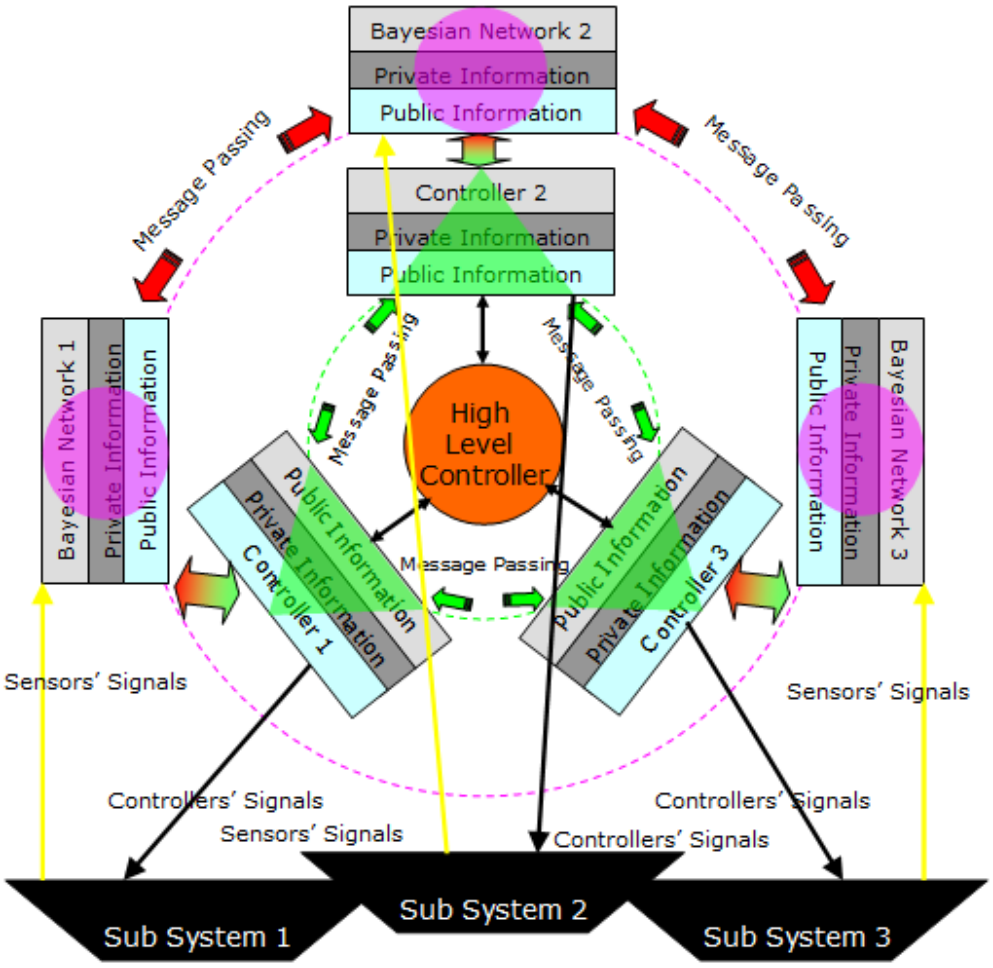


FIGURE 73 THE WHOLE CONTROL SYSTEM

5.2 Assumptions of Implementations

As the topic of this research indicates, the proposed methodology and process are for the control system design of a general large-scale complex system. In order to show the effectiveness of the methodology and process, and give the readers examples to look into, two implementations will be demonstrated step by step in the next chapter. However,

listing all of practical details of an application is far beyond an individual's possible workload. Herein, without deviating from the main theme of this research, a few assumptions are made about the implementations presented in the next chapter:

- Each agent in the whole system has a list of neighbor agents, and knows how and when to communicate with which agents during the control process. The list of neighbor agents is fixed when an agent is created. This is a reasonable assumption. Through broadcasting, registering functional roles, and carefully designing protocols, an agent can communicate with any agent in the system in a dynamic way. However, the "agent discovery" is itself a challenging task and needs a great deal of work from computer and communication experts. It is beyond the scope of this research.
- Communication channels are reliable and their capacity is not limited. There are many ways to improve reliability of communications. An agent communicates only with a limited number of neighbor agents. And current techniques about communication capacity are advanced enough for most system communication requirements. If the communication system is not reliable, DSTO, DDSSS, DRIS and DBP will be initiated automatically for the whole system. However, in the application described in the next chapter, only DBP is implemented while DSTO, DDSSS and DRIS are not triggered.
- The data processing time of individual agent is quick enough and can be neglected compared with the delay of the dynamic of the system itself. This assumption is reasonable because the whole system consists of a lot of agents and each agent processes a small part of the whole system and current unit processor is powerful enough.
- The fault type of components is discrete, i.e., there are no intermediate types of faults or failures. For example, a valve just has two damage states of either stuck open or

stuck closed, i.e., the inference does not distinguish between a valve's stuck openness at 30 percent or 40 percent of full openness.

- All of the states of a component are convertible with each other in order to avoid state estimation conflicts, i.e., there is no absorbing state for a component. By giving very small chance of unusual state exchanging, this assumption is reasonable. For example, when a valve is stuck open, if a close command comes, the state of the valve remaining stuck open is highly likely. However, the system still gives a very small chance that the state of the valve may become stuck closed or closed.
- The sensor data is contaminated with noises but with no other failures. However, according to the displayed value of a sensor, the likelihood of the true value is known. Improving sensor reliability and filtering sensed data with noises themselves are very challenging tasks.
- The structure of sub Bayesian networks is available and the prior probability parameters are also known. Bayesian network learning (structure learning and parameter learning) is still an active research area and efficient Bayesian network learning is itself a very challenging task. Collecting practical historical data, and learning Bayesian network structure and parameters need expertise's work in any specific application of Bayesian network inference, which is out of the scope of this research. The author is aware of that the structure and prior beliefs of a Bayesian network will highly likely affect the quality of the inference processing. In the following application, some simple but reasonable cause-effect relationships and prior distributions are used.

CHAPTER VI

APPLICATION

As mentioned in Chapter II, the control system design for a large-scale complex system is a substantial project which needs numerous people to work coordinately over a significant time span. Designing a comprehensive control system for a large-scale complex system down to detail is not the objective of this research. However, in order to check the effectiveness and validity of the proposed methodology and process and show how to implement the process step by step, in this chapter, the control system design of a simplified ship-wide Chilled Water System (CWS) of DDG51 will be given as a proof of concept.

6.1 Introduction

6.1.1 Automation of Ship Board Control System

Automation of ship board control system of Navy ships and submarines has been researched for a long time and it continues to be improved in order to meet the increase in war fighting capability and reductions in manning envisioned for the future fleet. More specifically, most of the research focuses on rapid resource reconfiguration, automation to reduce workload and the capability to incorporate high power weapons, sensors and new technologies in a more adaptable and efficient way.

In [14], an introduction to the evolution of ship board control systems from the 1970's to 2000's was given. In the 1970's, the techniques of computers and computer networks were first implemented in ship board control systems of DD963 class ships. In the late

1980's, a more extensive use of digital computers and data multiplex system communication networks were equipped on DDG51 class ships. In the 1990's, the on-board ship control system became even more distributed especially with input/output interfaces by using advanced network techniques. In the early 2000's, a highly distributed control system with limited use of smart sensors and actuators was implemented on DD21 class ships. Figure 74 shows a summary of this evolution of ship on-board automation and control systems.

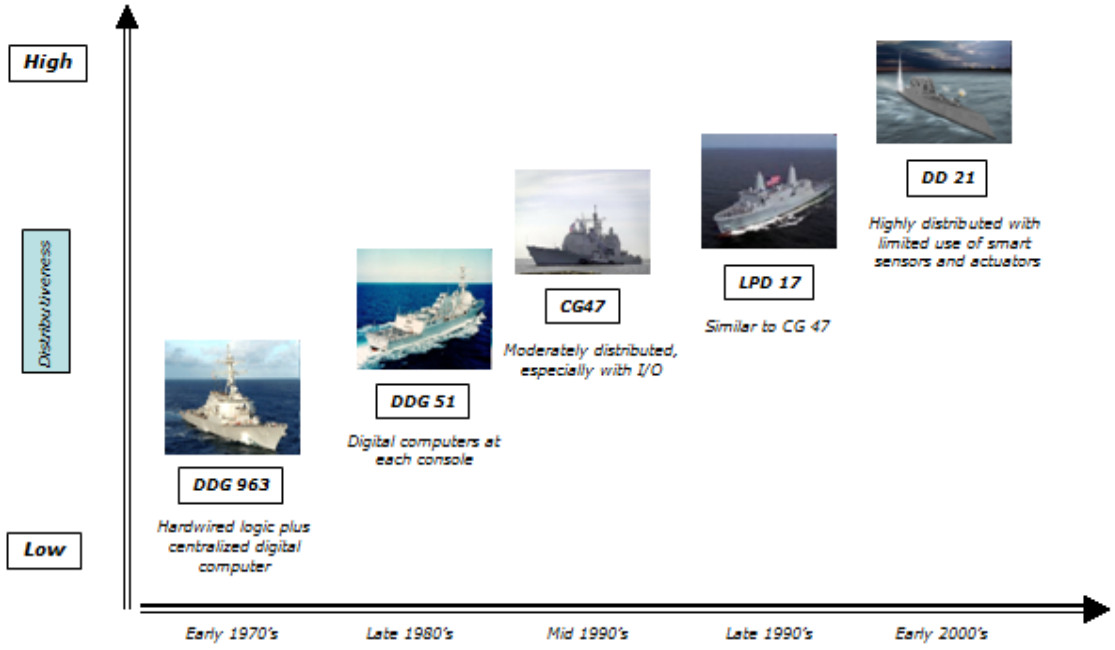


FIGURE 74 EVOLUTION OF ON-BOARD CONTROL SYSTEM OF NAVY SHIPS

As we can see from Figure 74, the development goals of ship on-board control systems are to be more distributed and intelligent by using more advanced communication network and digital processing techniques to

- increase combat effectiveness,
- increase survivability,

- and reduce total ownership cost.

6.1.2 Ship Board Chilled Water System

The chilled water distribution control system as an important part of the ship board control system plays a vital role in many cooling applications on ships and submarines for providing cooling water to:

- communication systems,
- radar,
- sonar,
- combat systems (military applications),
- ship control systems,
- cooling air for onboard air-conditioning systems to ensure a pleasant working environment for the crew and passengers,
- and other electronic equipments, etc.

The chilled water distribution system on a ship is a good example of a complex system. It is distributed over the entire ship and exhibits global behaviors. In addition, its environment is uncertain, especially under combat situations. It is almost impossible to predict all possible damage scenarios and derive and store the corresponding strategies in memory to reconfigure the system. Currently, this system is primarily operated and managed manually. Since manning is a considerable portion of the ownership cost of US Navy ships, manual control of these highly complex systems directly impacts the cost of supporting these assets in a significant manner. For this reason, automating the control of the chilled water system and empowering sailors to be able to operate it more efficiently is an important goal if one is to attempt to reduce ownership costs of Navy ships.

Automation on ship board control system especially using a distributed agent-based control concept for chilled water system has been researched and tested for a long time. Rockwell Automation (RA) is in the leading role in developing an autonomous cooperative system agent platform for controlling ship chilled water system. RA focuses on control distribution and automation, but not on uncertainty inference and control architecture development of general large-scale complex systems.

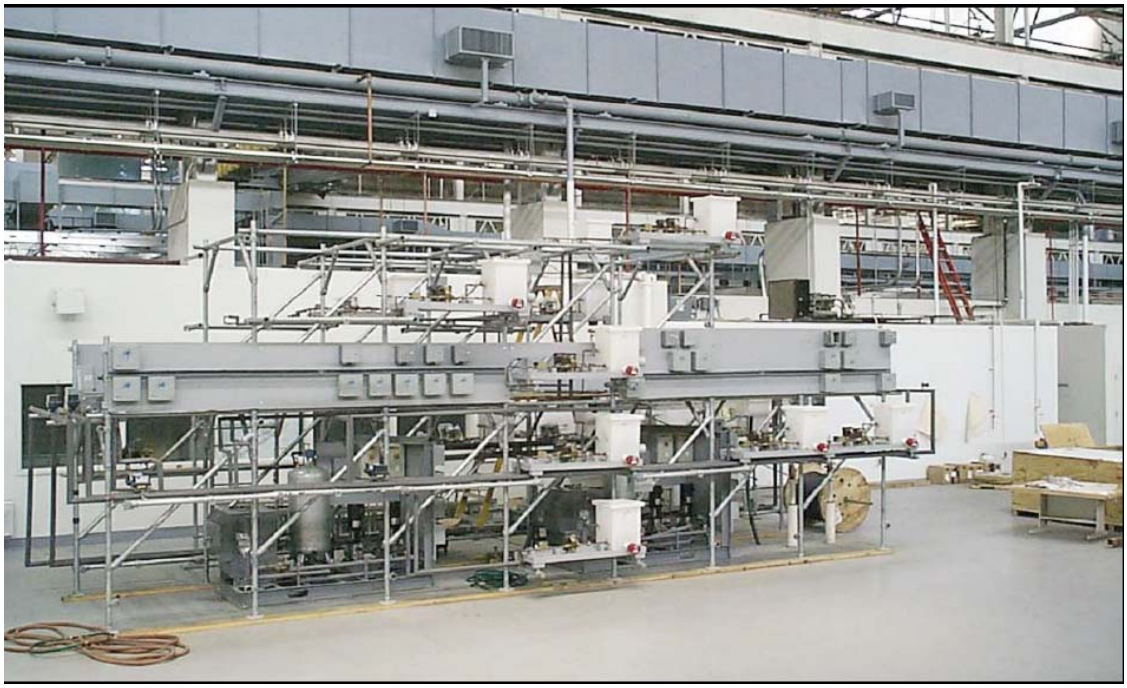


FIGURE 75 THE CHILLED WATER TEST BED [19]

In this chapter, control system design of a simplified Chilled Water System (CWS) based on the Reduced Scale Advanced Demonstrator (RSAD) model will be used as an application of the proposed methodology and process. CW-RSAD is a scaled-down version of a real ship chilled water system developed by the Naval Surface Warfare Center Carderock Division (NSWC-CD) in Philadelphia. The physical model of RSAD is shown in Figure 75. It contains two distributed chilled water resource centers and 16 service loads. Each resource center includes two pumps, 2 expansion tanks and one

chiller plant. The two pumps in one resource center parallel as back up for each other. In-line tanks containing controllable heaters simulate equipments as service loads which need to be cooled down directly by the chilled water system.

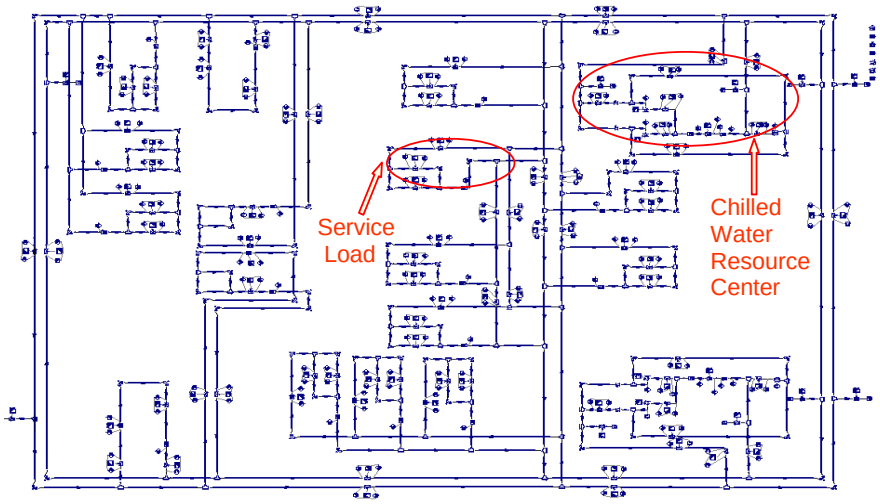


FIGURE 76 CWS-RSAD MODEL IN FLOWMASTER

Testing the control system directly by connecting to the physical system is beyond the scope of this research. Fortunately, a Flowmaster model of the CWS-RSAD system has been established by Naval Surface Warfare Center (NSWC). Flowmaster is a system simulation software tool used by companies across a wide range of industries to reduce the development time and costs of their thermo-fluid systems. Flowmaster can be used at concept formation phase of the development process, design and optimization phase and preliminary prototype validation phase, etc. In this application, by using a Flowmaster digital prototype/model, the control system can be tested through simulation in computers without the need of the physical piping systems, sensors and actuators of real control systems. A sketch of the Flowmaster model of the CWS-RSAD system is shown in Figure 76. A typical service load structure and one chilled water resource center are shown in Figure 77 and Figure 78 respectively.

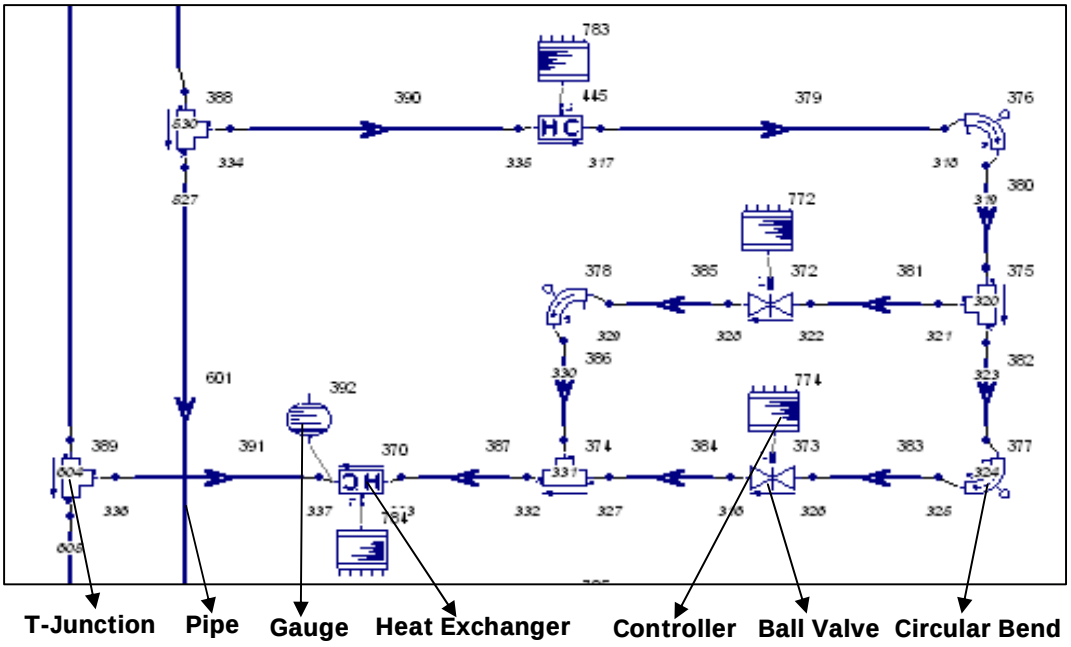


FIGURE 77 STRUCTURE OF A TYPICAL SERVICE LOAD

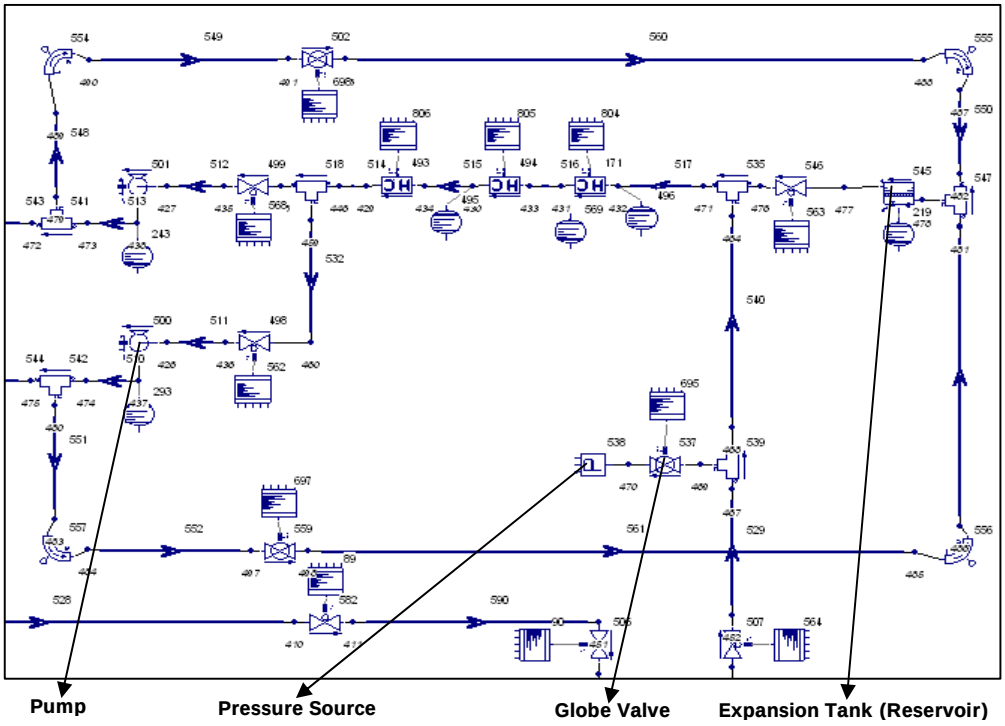


FIGURE 78 STRUCTURE OF CHILLED WATER RESOURCE CENTER

In this chapter, a further simplified chilled water system shown in Figure 79 will be used to show steps of the proposed process.

FIGURE 79 A SIMPLIFIED SHIP CHILLED WATER SYSTEM

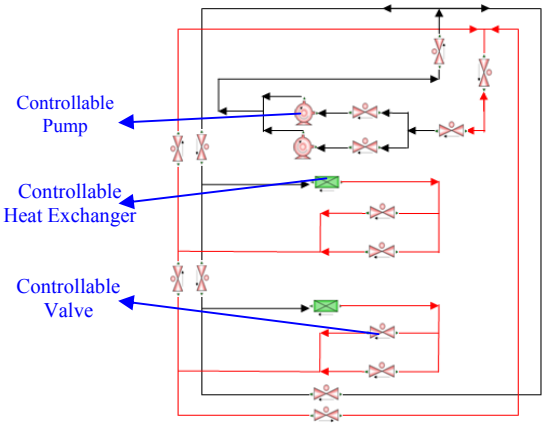


FIGURE 80 A CLEAR SKETCH OF THE SIMPLIFIED SHIP CHILLED WATER SYSTEM

By deleting the controller icons, flow meter icons, component indexes, etc. in the corresponding Flowmaster model, a clearer sketch of the simplified chilled water system is shown in Figure 80. The black lines indicate pipes with lower temperature cooling water and the red lines indicate pipes with higher temperature return water. In this model, only one chilled water resource center and two service loads are included. The chilled water resource center structure and the two service loads internal structures are as the same as those in the original comprehensive CWS-RSAD model. Detailed descriptions about how to establish a multi-agent based control system with a distributed Bayesian network inference engine will be given in the following sections.

6.2 Control System Design of the Simplified CWS-RSAD Model

In this section, a control system for the simplified CWS-RSAD model will be established step by step according to the proposed methodology and process in this research.

6.2.1 Smart Valves

The first smart valve was introduced more than a decade ago with the introduction of its integrated control loop. With the advancement of networking technology and continuing

improvement of unit processor, sensor and servo techniques, more new products with smart valves have been developed with the expectation of much greater performance and provide distributed and intelligent control in many related applications [118, 119].

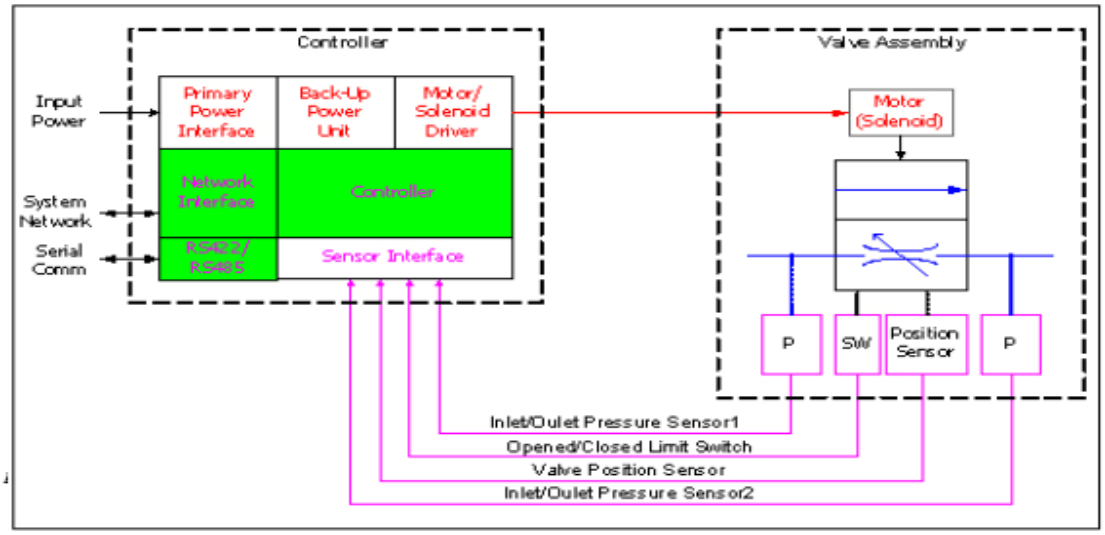


FIGURE 81 FUNCTIONS OF SMART CONTROL VALVE DESIGN [119]

A smart valve consists of two parts: intelligent fluid controller and valve assembly. The intelligent fluid controller includes embedded processor, sensor data acquisition interfaces, network interfaces and actuator output signal interfaces. The valve assembly includes actuators, sensors and a conventional valve itself. Figure 81 shows function integration of intelligent chilled water valve Marotta Model MV 286S. MV 286S had been installed at the Land Based Test Site at Naval Surface Warfare Center, Philadelphia. With a low power requirement and emergency backup power, MV 286S also allows deployment in applications that are not suited to conventional valves, where standard power sources are not available or are unreliable.

A smart valve can be connected to a data network through network interfaces, so all available advantages in computer networks can be attained. Network connectivity allows for distributed control, by virtue of strategically placed control nodes that control and

monitor many such smart valves, such as real time fault detection and isolation and system level resource allocation and reconfiguration as intelligent responses to some damage conditions or new environment conditions.

A smart valve has its own processor, lower power requirements, and emergency backup power. The intelligent controller mounted on a valve and linked to sensors and actuators directly creates an easy local control loop with enhanced communication capabilities. It can work as standalone device with no additional infrastructure when it is completely cut from the network due to some network damages [120].

6.2.2 Smart Pumps

As in a smart valve, a smart pumping system includes an intelligent fluid controller and a pump assembly. The intelligent fluid controller consists of a micro processor, sensor data acquisition interfaces, network interfaces and actuator output signal interfaces. The pump assembly includes standard pump, actuators and sensors, etc. Such an integrated pumping system can achieve certain objectives automatically, locally and independently. For example, a centrifugal pump with a variable speed drive, operating costs could be reduced significantly by eliminating the pressure drop across a control valve. Fault tolerance could also be built into the pumping system by developing special software that would interact with instrumentation signals that sense process conditions. The software would require the ability to recognize and prevent the pump from operating under damaging conditions. Finally, if a method could be developed to use the pump casing as a flow-measuring device; in many applications the need for a separate flow meter would be eliminated [121]. Similarly, a smart pumping system can be connected to a data network though network interfaces and has the capability of network coordinated control.

Smart valves and smart pumping systems provide the foundation of distributed control in the chilled water system.

6.2.3 Thermoelectric Model

For an electrical device, part of the power supplied by the power system will dissipate into heat energy due to resistance. If the heat energy can not be removed, the component temperature will climb to be out of its operation temperature limit and the component will be damaged. In a navy ship system, power component heat dissipation is removed through conduction from the power component to a "cold plate" heat sink. This cold plate is then cooled via convective heat transfer to cooling fresh water. The cooling fresh water is cooled down by an open loop sea water cooling system. The whole cooling system for one service load is shown in Figure 82. The model in this dissertation will assume that the sea water cooling system is always working by fixing the temperature of fresh water out of the sea water cooling system. This assumption is not practical in the real situation. However, it will not effect the evaluation of the proposed methodology in this dissertation. In an application in the future, the sea water cooling system could be added as another subsystem and one/a few Bayesian network agents and control agents could be added through extension of the system established in this dissertation.

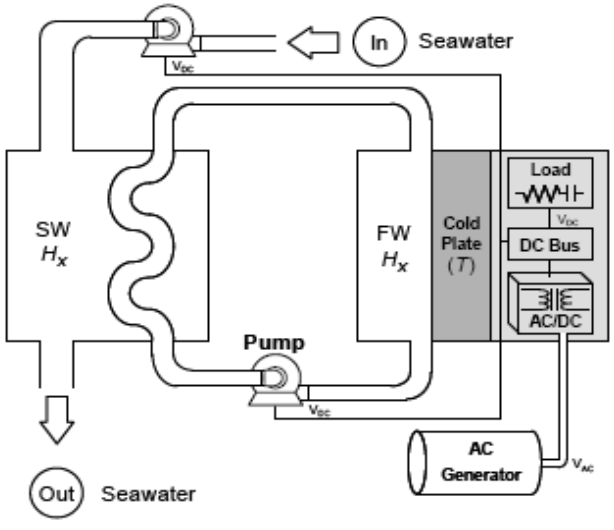


FIGURE 82 PHYSICAL REPRESENTATION OF SEA WATER, FRESH WATER, COLD PLATE AND ELECTRICAL DEVICE HEAT EXCHANGERS [122]

The model of the fresh water cooling system is shown in Figure 83. In this model, we assume that the heat extracted by ambient air is far less than the heat extracted by the fresh water and can be neglected. The following equations are excerpted from a paper written by Lee et al [123] . Table 4 lists all of the parameters involved in the following equations and their corresponding units used in the simulations of this dissertation.

TABLE 4 LIST OF TERMS IN THERMOELECTRIC MODEL

Terms	Description	Units
P_{comp}	Component power	W
η_{comp}	Power component efficiency	N/A
Q_{in}	Heat transfer rate from the component	W
w_{cf}	Mass flow rate of the cooling fluid	kg/s
T_{ci}	Temperature of the incoming fluid	K
T_{co}	Temperature of the outgoing fluid	K
T_{hx}	Temperature of the cold plate	K
m_{hx}	Mass of the cold plate	kg
c_{hx}	Specific heat of the cold plate	$J/kg/K$
c_{cf}	Specific heat of the cooling fluid	$J/kg/K$
A	Cooling fluid contact area	m^2
h	Cooling fluid heat transfer coefficient	$W/m^2/K$
Q_{hx}	Heat transfer rate from the cold plate to fluid	W
$\dot{T}_{hx}$	Time derivative of the temperature of the cold plate	kg/s
k_p	Proportional control coefficient	N/A
$T_{co\_threshold}$	The upper limit out flow temperature at which the cooling fluid heat removal capacity is saturated	K
T_{ctrl}	The control temperature (the component temperature above which cooling fluid is required)	K
T_{hx_ul}	The upper limit of a semiconductor power device normal operation	K

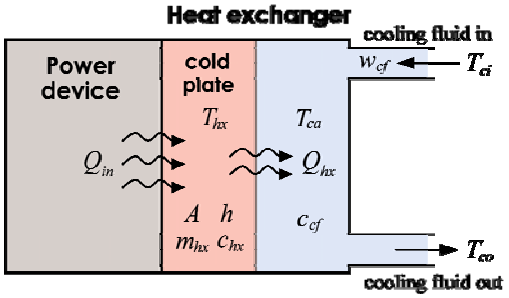


FIGURE 83 POWER DEVICE, COLD PLATE AND FLUID HEAT EXCHANGER [123]

The power component heat dissipation is determined from the power component efficiency:

$$Q_{in} = (1 - \eta_{comp}) P_{comp} \tag{6.1}$$

Note that in this formulation, Q_{in} is independent of the cold plate temperature and the thermal resistance between the power components and the cold plate. These assumptions are consistent with the customary deactivation of power components from excessive cold plate temperature. The heat exchanger is assumed to be well insulated and the fluid flow is assumed to be one dimensional flow.

The net heat flow into the heat exchanger cold plate is the difference between the heat flow from the power components and the heat flow removed by the cooling fluid. This net heat flow determines the cold plate rate of temperature change:

$$\dot{T}_{hx} = \frac{Q_{in} - Q_{hx}}{m_{hx} c_{hx}} \tag{6.2}$$

The heat flow from the cold plate to the cooling fluid is governed by the convective heat transfer from the cold plate to the cooling fluid:

$$Q_{hx} = Ah(T_{hx} - T_{ca}) \tag{6.3}$$

From conservation of energy, the heat flow removed by the cooling fluid can be specified in terms of flow rate, specific heat and fluid heating:

$$Q_{out} = w_{cf} c_{cf} (T_{co} - T_{ci}) \tag{6.4}$$

Neglecting transient fluid thermodynamics along the interior of the heat exchanger produces a lumped parameter fluid model wherein $Q_{out} \approx Q_{hx}$ and the average fluid temperature:

$$T_{ca} = \frac{(T_{ci} + T_{co})}{2} \tag{6.5}$$

Substituting equation (6.5) into equation (6.3) provides the heat transfer rate of cold plate to cooling fluid in terms of inlet and outlet fluid temperatures:

$$Q_{hx} = Ah\left(T_{hx} - \frac{T_{ci} + T_{co}}{2} \right) \tag{6.6}$$

Rearranging equation (6.6) yields the outlet temperature of the cooling fluid:

$$T_{co} = 2\left(T_{hx} - \frac{Q_{hx}}{Ah} \right) - T_{ci} \tag{6.7}$$

Using the $Q_{out} \approx Q_{hx}$ approximation, equation (6.4)can be rearranged, so that

$$T_{co} = T_{ci} + \frac{Q_{hx}}{w_{cf} c_{cf}} \tag{6.8}$$

Substituting equation (6.8) into equation (6.6) yields the heat transfer rate in terms of the cold plate temperature, inlet cooling fluid temperature and flow rate:

$$Q_{hx} = \frac{2 w_{cf} c_{cf} Ah}{2 w_{cf} c_{cf} + Ah} \left(T_{hx} - T_{ci} \right) \tag{6.9}$$

However, this formulation is only valid if the cooling fluid flow rate is sufficient so that the heat transfer rate to the fluid Q_{hx} does not exceed the cooling fluid heat removal capacity defined by constraining equation (6.9) and $T_{co} \le T_{hx}$. If the cooling fluid heat removal capacity is saturated, then $T_{co} \approx T_{hx}$ and from equation (6.4), get

$$Q_{hx} \approx Q_{out} = w_{cf} c_{cf} (T_{hx} - T_{ci}) \tag{6.10}$$

Comparing equation (6.6) and equation (6.9) at the $T_{co} = T_{hx}$ boundary condition provides a convenient test to ensure that the heat transfer to the fluid does not exceed the heat removal capacity of the fluid:

$$2 w_{cf} c_{cf} \ge Ah \tag{6.11}$$

In summary, equations (6.2), (6.7), (6.9) and (6.11) provide the final cold plate heat exchanger thermal model including thermal saturation:

$$\dot{T}_{hx} = \frac{Q_{in} - Q_{hx}}{m_{hx} c_{hx}}$$

where (6.12)

$$\begin{cases} Q_{hx} = \dfrac{2 w_{cf} c_{cf} Ah}{2 w_{cf} c_{cf} + Ah} (T_{hx} - T_{ci}) \quad \& \quad T_{co} = T_{ci} + \dfrac{Q_{hx}}{w_{cf} c_{cf}}, & \text{if } 2 w_{cf} c_{cf} \ge Ah \\ Q_{hx} \approx Q_{out} = w_{cf} c_{cf} (T_{hx} - T_{ci}) \quad \& \quad T_{co} = T_{hx}, & \text{if } 2 w_{cf} c_{cf} < Ah \end{cases}$$

In this dissertation, a simple proportional feed back control shown in (6.13) is used to adjust the cooling resource requirement for power component:

$$\left(T_{hx} - T_{ctrl}\right)' = -k_p (T_{hx} - T_{ctrl}) \tag{6.13}$$

where k_p is the proportional control coefficient and it effects how fast a power component will be cooled down. Based on equation (6.13) and the condition whether the

cooling fluid heat removal capacity is saturated or not, the process of calculation of the required cooling fluid flow rate and the cooling fluid outgoing flow temperature are as follows:

$$Q_{hx} = m_{hx} c_{hx} k_p (T_{hx} - T_{ctrl}) = \frac{2 w_{cf} c_{cf} Ah}{2 w_{cf} c_{cf} + Ah} (T_{hx} - T_{ci})$$

$$\Rightarrow \begin{cases} w_{cf} = \dfrac{1}{2 c_{cf} \left(\dfrac{T_{hx} - T_{ci}}{m_{hx} c_{hx} k_p (T_{hx} - T_{ctrl})} - \dfrac{1}{Ah} \right)} \\ T_{co} = T_{ci} + \dfrac{Q_{hx}}{w_{cf} c_{cf}} \end{cases} \tag{6.14}$$

$$w_{cf} c_{cf} (T_{hx} - T_{ci}) = Q_{in} + m_{hx} c_{hx} k_p (T_{hx} - T_{ctrl})$$

$$\Rightarrow \begin{cases} w_{cf} = \dfrac{Q_{in} + m_{hx} c_{hx} k_p (T_{hx} - T_{ctrl})}{c_{cf} (T_{hx} - T_{ci})} \\ T_{co} = T_{hx} \end{cases} \tag{6.15}$$

$$w_{cf} c_{cf} (T_{co} - T_{ci}) = Q_{in} + m_{hx} c_{hx} k_p (T_{hx} - T_{ctrl})$$

$$\Rightarrow \begin{cases} T_{co} = T_{co\_threshold} \\ w_{cf} = \dfrac{Q_{in} + m_{hx} c_{hx} k_p (T_{hx} - T_{ctrl})}{c_{cf} (T_{co} - T_{ci})} \end{cases} \tag{6.16}$$

Calculate Required w_{cf} ***and*** T_{co}

if $T_{hx} < T_{co\_threshhold}$
  if $T_{co} <= T_{hx}$
    calculate w_{cf} *and* T_{co} *according to* (6.14)
  else
    calculate w_{cf} *and* T_{co} *according to* (6.15)
  end if
else
  if $T_{co} <= T_{hx}$
    calculate w_{cf} *and* T_{co} *according to* (6.14)
  else

calculate w_{cf} *and* T_{co} *according to* (6.16)

end if

end if

6.2.4 Implementation of the Proposed Methodology to the Simplified CWS-RSAD

6.2.4.1 Step 1: Check System Constraints & Requirements

The general objective of the chilled water system is to provide continuous chilled water through pipelines to the heated service loads to keep their operation temperatures under specified values. The maximum chilled water flow rate depends on the speed of the pumps. One time, only one pump is working and the pump speed can not exceed its upper limit. Only a subset of all of the flow rates are measurable. All of the valves and pumps are shown in Figure 80 and we can see the main circulation piping is looped to provide alternative paths from the chilled water resource to either service load as a way to increase the system survivability. However, the control system needs to have the capabilities to detect component failures and diagnose which routes are available and to pick different routes based on resource capability, service load requirements, service load priorities, route states, etc.

In summary, the control system needs to satisfy the following system level requirements:

- Automation: the system can run with less human intervention.
- Robust: the system can run when the information for the control system is contaminated.
- Reconfiguration: the system can reconfigure itself when parts of the system are damaged.
- Dynamics and Flexibilities: the system can adapt to changes of the environment over time.

6.2.4.2 Step 2: Decompose System

The system is decomposed hierarchically into module combinations. The entire system as the high level is decomposed into three subsystems as shown in Figure 84: chilled water resource center, service load 1 and service load 2. The chilled water resource center consists of 5 valves and two pumps; service load 1, as well as service load 2, contains one heat exchanger and two parallel valves; also three pairs of valves (cross valves) are provided to connect the chilled water resource center and the service loads. Those cross valves can also be taken as a subsystem and they can be further decomposed into two groups: one group contains cross valves of getting chilled water and one group contains cross valves of sending hot water back. How to combine components at the same level into different modules will be shown in step 3 and step 4.

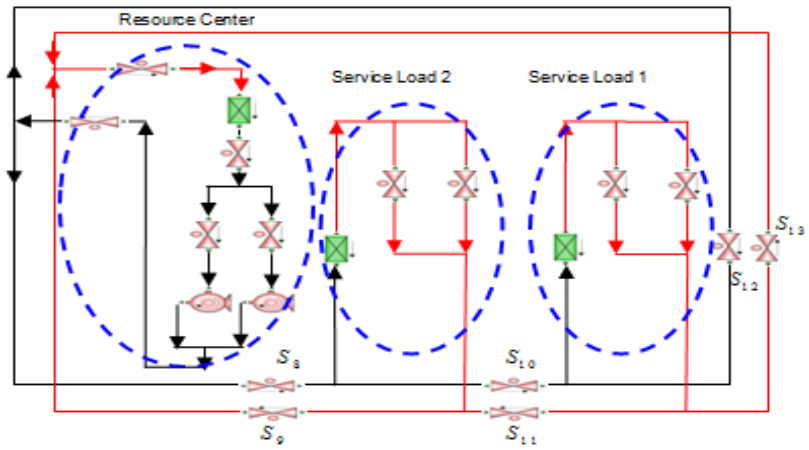


FIGURE 84 DECOMPOSING THE SIMPLIFIED CHILLED WATER SYSTEM

There are multiple routes for each service load to get chilled water from the resource center and to send hot water back to the resource center. For example, chilled water to service load 1 can be controlled using valves S_8, S_{10} or S_{12}; chilled water for service load 2 can be controlled using valve S_8, S_{10} or S_{12}. Similarly, hot water back to the chilled water resource center from service load 1 can be controlled using S_{11}, S_{13} or S_9; hot

water back to the chilled water resource center from service load 2 can be controlled using valves S_9, S_{11} or S_{13}.

6.2.4.3 Step 3: Establish Control Architecture and Control agents

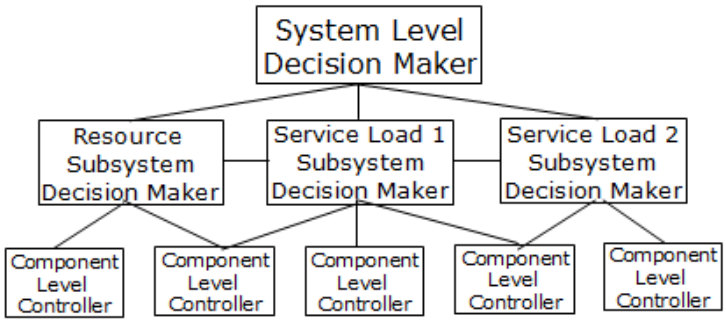


FIGURE 85 CONTROL ARCHITECTURE OF THE SIMPLIFIED CHILLED WATER SYSTEM

According to the decomposition, the whole system has three control levels as shown in Figure 85: system level decision maker, subsystem decision makers (resource subsystem decision maker, service load 1 subsystem decision maker, service load 2 subsystem decision maker) and component level controllers. The system level agent, subsystem agents and component agents as the three main types of agents have different internal logic and objectives. The internal logic will be implemented as a series of behaviors of an agent in JADE.

The system level agent collects information from subsystem agents and gives high level commands to subsystem agents without giving detailed commands down to the components. For example, a resource subsystem agent sends the state of the whole resource center (opened, closed or damaged) to the system level agent; it will not point out which components in resource center are opened, closed or damaged. Similarly, a system level agent will only give open, close or no action to the whole resource center; the resource center will implement this command through control of its components. In

order to improve the survivability of the system, a system level agent, as a critical agent, has several replications that are organized into a ring structure.

Subsystem level agents send their system states to a high level agent, get commands from a high level agent and try to implement those commands by controlling their corresponding components. For example, initially, all of the components in resource center are undamaged and closed, so it sends a "Close" state to the high level agent. The high level agent will give an "Open' command to the resource center. Once the resource center gets the "Open" command, it will give different commands to its corresponding components, such as "Open" commands to all of the components except "Close" commands to one pump and one valve following this pump. Similarly, initially, for a service load, one of the parallel valves will get the command of "Open" and the other will get the command of "Close". Subsystem level agents are important as well, so replication rings could be used. In addition, direct communications among the subsystem level agents can be used to further improve the robustness of the whole system.

Component level agents are the lowest level agents in this hierarchical control system and they interact with their subsystem level agents and the actuators (here, a Flowmaster model is used) directly. In this implementation, there are three types of component agents: valve, pump and heat exchanger. In order to be flexible, the state of those components are normalized, standardized and share the same class file in Java. For example, the valve, the pump and the heat exchanger have the same discrete state space $\{Open, Close, StuckOpen, StuckClose\}$. "StuckOpen" means this component is fully working, but it can not be shut down. "StuckClose" means this component is closed and can not be opened. Such an assumption of four states is not perfect, but reasonable in a simplified situation. For a real mechanical system, usually, the chance of a component stuck in the middle is very small. For more accurate modeling, the state space can be extended.

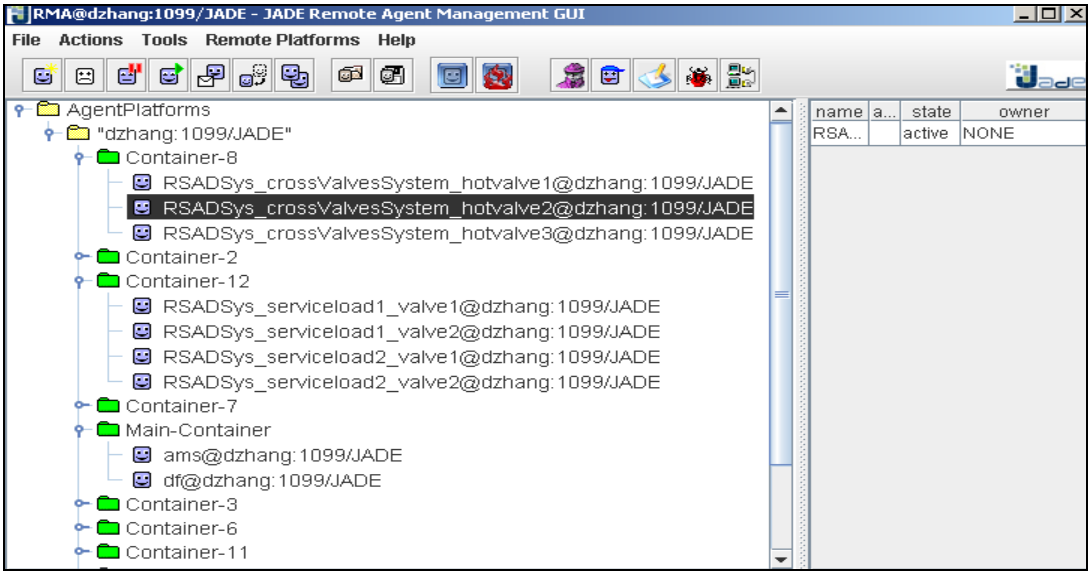


FIGURE 86 AGENTS OF THE SIMPLIFIED CHILLED WATER CONTROL SYSTEM ESTABLISHED IN JADE

All of the agents established in JADE for the simplified chilled water control system are shown in Figure 86 and the detailed code in Java is provided in Appendix E.

6.2.4.4 Step 4: Establish Sub System Bayesian Networks

In this step, three subsystem Bayesian networks will be established: resource center sub Bayesian network, service load 1 sub Bayesian network and service load 2 sub Bayesian network.

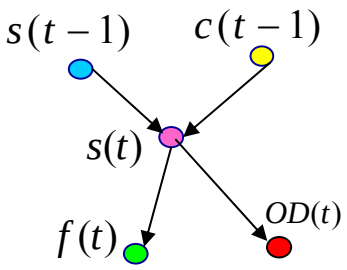


FIGURE 87 TWO-TIME SLICE HOMOGENEOUS DYNAMIC BAYESIAN NETWORK FOR ONE COMPONENT

Herein, two-time slice homogeneous DBN will be used. The future state of a component is conditionally independent on its past states and past commands if the current state and current command are specified. A sketch of this idea is shown in Figure 87. $s(t-1)$ is the component state at time $t-1$ and it has four discrete states as described in the previous section. $c(t-1)$ is the component command at time $t-1$ and it has two discrete states as described in the previous section. $OD(t)$ is the component open degree at time t and it has two discrete states $\{Open, Close\}$. $f(t)$ is a possibly measurable flow rate point. However, in reality, it may be not observable. It has three states $\{NegativeFlow, PositiveFlow, NoneFlow\}$. The positive flow direction is specified for each measurable flow rate point at the beginning of simulation. For a Bayesian network, variable prior distributions and conditional distributions are important parts that are vital for the inference accuracy. Normally, this information as well as the cause-effect relationships are determined from experts in those specific fields, or extracted through some modeling techniques from existing data sets. However, in this application, those data are not available and we will presume those prior distributions, conditional distributions and cause-effect relationships are known.

The prior distribution of a component state and a component command are represented as potentials as shown in Table 5 and Table 6 respectively. The transition condition probability distribution represented as potentials is shown in Table 7. Those distributions are notional, but reasonable. Other prior distribution and conditional distribution tables are shown in Appendix G.

TABLE 5 COMPONENT STATE PRIOR DISTRIBUTION

Component State: $s(t_0)$	Index	Potential
Open	1	0.25
Close	2	0.25
StuckOpen	3	0.25
StuckClose	4	0.25

TABLE 6 COMPONENT COMMAND PRIOR DISTRIBUTION

Component Command $c(t_0)$	Index	Probability
Open	1	0.5
Close	2	0.5

TABLE 7 TRANSITION CONDITIONAL STATE DISTRIBUTION OF A COMPONENT

No.	Component State $s(t-1)$	Component Command $c(t-1)$	Component State $s(t)$	Potential
1	Open	Open	Open	1
2	Close	Open	Open	1
3	StuckOpen	Open	Open	0.000001
4	StuckClose	Open	Open	0.000001
5	Open	Close	Open	0.000001
6	Close	Close	Open	0.000001
7	StuckOpen	Close	Open	0.000001
8	StuckClose	Close	Open	0.000001
9	Open	Open	Close	0.000001
10	Close	Open	Close	0.000001
11	StuckOpen	Open	Close	0.000001
12	StuckClose	Open	Close	0.000001
13	Open	Close	Close	1
14	Close	Close	Close	1
15	StuckOpen	Close	Close	0.000001
16	StuckClose	Close	Close	0.000001
17	Open	Open	StuckOpen	0.000001
18	Close	Open	StuckOpen	0.000001
19	StuckOpen	Open	StuckOpen	1
20	StuckClose	Open	StuckOpen	0.000001
21	Open	Close	StuckOpen	0.000001
22	Close	Close	StuckOpen	0.000001
23	StuckOpen	Close	StuckOpen	1
24	StuckClose	Close	StuckOpen	0.000001
25	Open	Open	StuckClose	0.000001
26	Close	Open	StuckClose	0.000001
27	StuckOpen	Open	StuckClose	0.000001
28	StuckClose	Open	StuckClose	1
29	Open	Close	StuckClose	0.000001
30	Close	Close	StuckClose	0.000001
31	StuckOpen	Close	StuckClose	0.000001
32	StuckClose	Close	StuckClose	1
33	Open	Open	Open	1

TABLE 7 CONTINUES

34	StuckClose	Open	Open	0.000001
35	Open	Close	Open	0.000001
36	Close	Close	Open	0.000001
37	StuckOpen	Close	Open	0.000001
38	StuckClose	Close	Open	0.000001
39	Open	Open	Close	0.000001
40	Close	Open	Close	0.000001
43	StuckOpen	Open	Close	0.000001
44	StuckClose	Open	Close	0.000001
45	Open	Close	Close	1
46	Close	Close	Close	1
47	StuckOpen	Close	Close	0.000001
48	StuckClose	Close	Close	0.000001

All measurable flow rate points in the simplified chilled water system are shown in Figure 88. As mentioned before, not all of the flow rate points indicated in Figure 88 are observable due to cost or head loss of flow meters. There are two general types of flow meters: invasive flow meter (with head loss), such as flow nozzle meter and non-invasive flow meter (without head loss), such as ultrasonic flow meters. Flow meters are either expensive or produce significant head loss. Therefore, in an application, the usage of flow meters is limited. The resulting resource center sub Bayesian network and service load sub Bayesian networks are shown in Figure 89 and Figure 90 respectively.

Herein, all of the DAGs are created from two time slices $t-1$ and t, based on the Markovia property. It consists of part of the nodes in time slice $t-1$ and all nodes in time slice t. The nodes in time slice $t-1$ only contain the nodes that have outgoing edges to time slice t.

Initially, the nodes in time slice t_0 with outgoing edges to time slice t_1 have uniform distributions. At time slice t, for every node with index $t-1$, it comes as soft evidence as $P(v_i(t-1) \mid y(t-1))$, which simplies the updating algorithms for DBNs. In this application, the control system only needs the current state estimation and it does not

need state smoothing and prediction, which need more complex and time consuming algorithms, such as frontier algorithm, interface algorithm [93], etc. Herein, we only use a simplified forward pass algorithm from interface algorithm of junction tree in general homogenous two-time slice DBNs developed by Murphy [93].

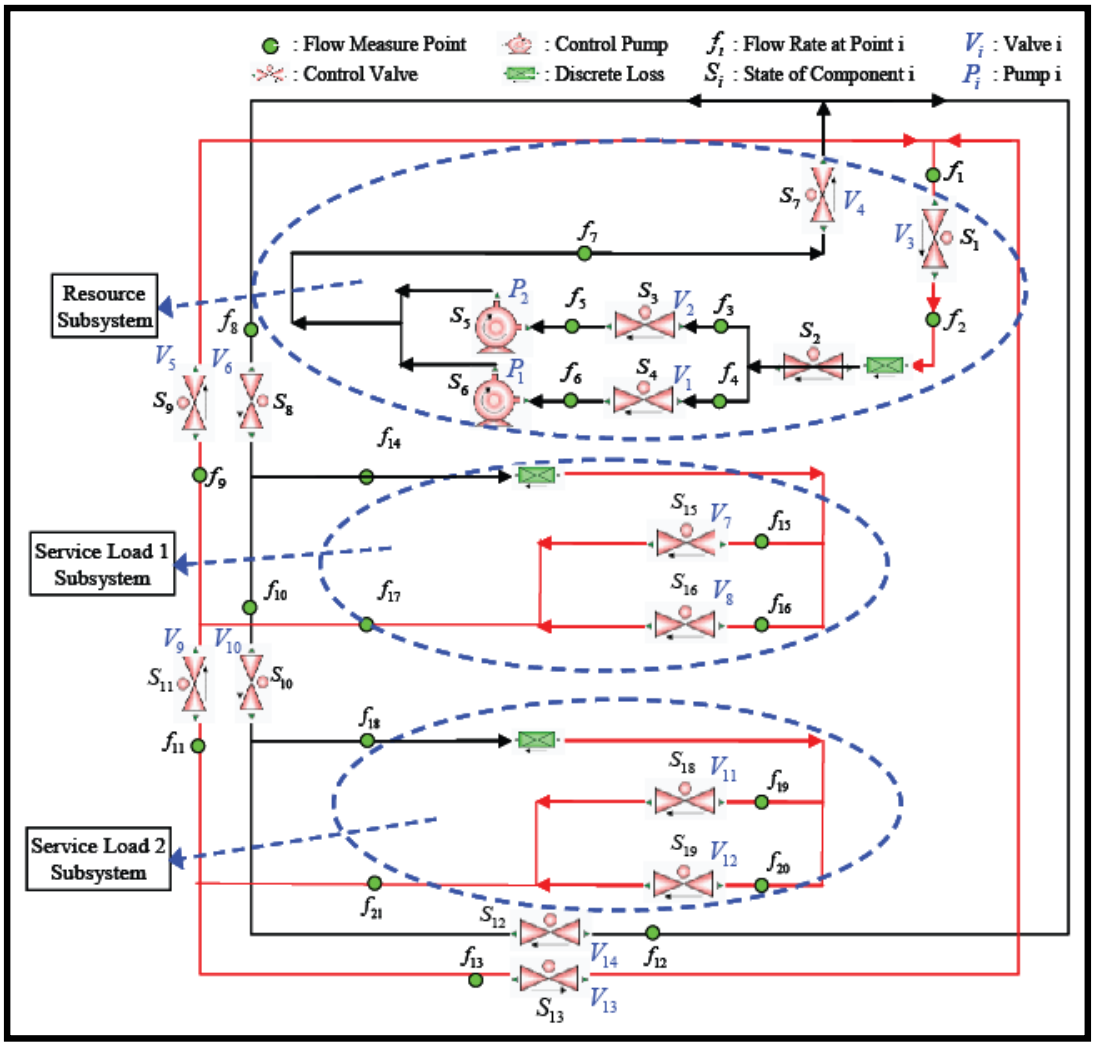


FIGURE 88 NODES IN BNS OF THE SIMPLIFIED CHILLED WATER SYSTEM

We assume the prior distribution for each node in the interface set $I(t_0)$ is a uniform distribution. All of the information from the previous time step will be added as soft evidence to current belief updating. This soft evidence timing the prior distribution of

uniform distribution is equal to the prior distribution, and it is the same result as setting the prior distribution as the soft evidence. In this way, previous time information would be easily fused into current time state inferences in Java Language by calling BayesNet functions. At time slice t, get each individual node distribution in the interface set $I(t)$, and then pass that distribution as soft evidence to next time slice $t+1$, which is called "fully factorized" approximation [93]. It is the most aggressive approximation of Boyen-Koller (BK) algorithm for two-time slice DBNs belief updating by assuming that the joint distribution of all of the nodes in the $I(t-1)$ is equal to the multiplication of each individual node's distribution. The basic idea of the BK algorithm is to approximate the joint distribution as a product of marginals, $P(I_t \mid y_{1:t}) \approx \prod_{i=1}^{k} P(I_t^i \mid y_{1:t})$,where I_t^i is the joint distribution on nodes in cluster i. The performance of BK algorithm depends on the clusters that we use to approximate the joint distribution of the interface nodes. In the application for this dissertation, the dependency of two individual component states is weak, therefore, "fully factorized" approximation works well. The DBN belief updating process described previous is formalized as follows:

- Construct DAGs contains only nodes in time slice $t-1$, with outgoing edges to time slice t and all of the nodes in time slice t.

- Set the prior distribution of each node in the interface set $I(t-1)$ as a uniform distribution.

- Accept the distribution of each node in the interface set $I(t-1)$ as soft evidence.

- According to the observations in time slice t, use regular j-tree belief updating to get inference for each individual node in $I(t)$, which is $p(V_i \mid y_{1:t}, V_i \in I(t))$; pass that to next time step.

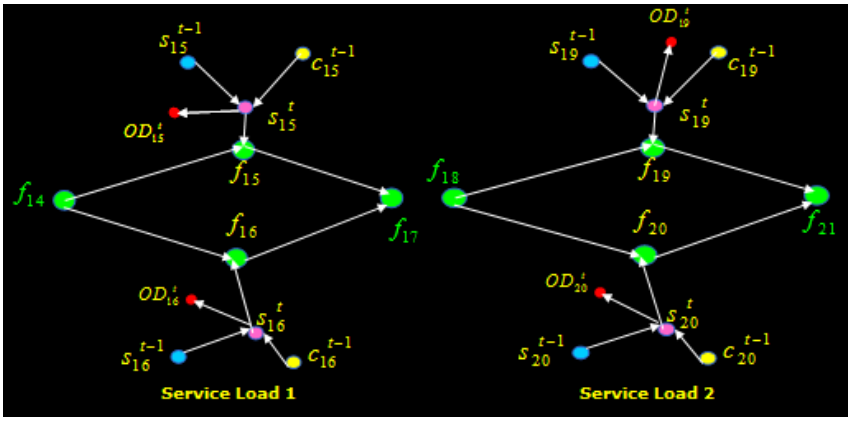


FIGURE 89 SERVICE LOAD SUB BAYESIAN NETWORKS

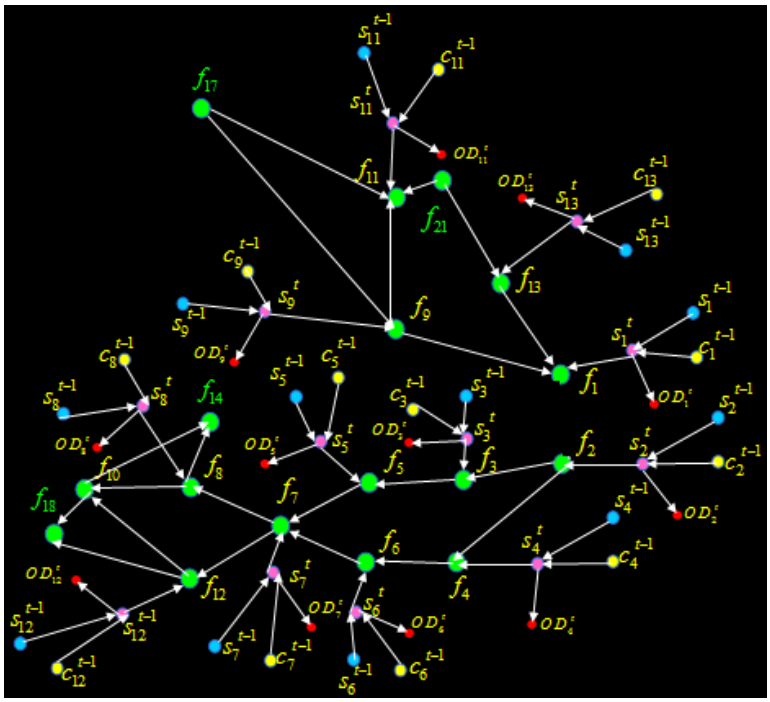


FIGURE 90 RESOURCE CENTER SUB BAYESIAN NETWORK

6.2.4.5 Step 5: Design Each Sub Bayesian Network as an Agent and Design Its Corresponding Internal Logic and Series of Behaviors for DSTO, DDSSS, DRIS and DBP

As described in Chapter IV, MSDBNs need to satisfy three conditions to get globally consistent inferences:

- tree organization,
- d-sep set between two neighbored sub Bayesian networks,
- and the running intersection property.

Automatic on-line formulation algorithms for these three conditions can be accomplished by using the three distributed algorithms described in Chapter IV. For this application, pre compilation is used instead of automatic on-line formulation for the three conditions. Only the internal logic and series of behaviors for DBP are implemented in each sub Bayesian network agent. A sub Bayesian network implementing DBP algorithm has the following general behaviors: *AcceptMessagesForCoTriangulation*, *AcceptCoTriangulationOverMessage*, *AcceptEvidenceFromComponent*, *AcceptEvidenceFromSensors*, *AcceptMessagsForCoBeliefUpdate,* and *AcceptMessagesForCoTriangulation*. *AcceptCoTriangulationOverMessage* is further broken down into three types of sub behaviors: *DepthFirstEliminate*, *FillInsFeedBack*, and *DistributeDLink*. *AcceptMessagesForCoBeliefUpdate* is also further broken down into four types of sub behaviors: *CollectBelief*, *AbsorbThroughLinkage*, *UpdateBelief*, *DistributeBelief*. Java source code describing the internal logic and series behaviors of each sub Bayesian network is presented in Appendix E.

FullBNT, an open source Bayes Network Toolbox written in MATLAB, is used to do individual sub Bayesian network modeling. The functions in MATLAB are called from Java by using JMatLink. JMatLink is an open source Java package that connects Java to MATLAB using Java Native Interface (JNI). Detailed descriptions of FULLBNT and JMatLink are shown in Appendix A and Appendix B respectively.

There are three sub Bayesian networks in this application. The service load 1 sub Bayesian network shares nodes f_{14} and f_{17} with the resource center sub Bayesian network; service load 2 shares nodes f_{18} and f_{21} with the resource center sub Bayesian

network. The organization of these three sub Bayesian networks is illustrated in Figure 91. The organization of the three sub Bayesian networks forms a simple tree structure – a chain, which satisfies the running intersection property. The parent nodes of node f_{14} are f_8 and f_{10}, which are in the resource center sub Bayesian network; parent nodes of node f_{17} are f_{15} and f_{16}, which are in service load 1 sub Bayesian network. Similarly, parent nodes of node f_{18} are f_{10} and f_{12} are in the resource center sub Bayesian network; parent nodes of node f_{21} are f_{19} and f_{20} are in service load 1 sub Bayesian network. According to the definition of d-sep set, set $\{f_{14}, f_{17}\}$ and $\{f_{18}, f_{21}\}$ are two d-sep sets.

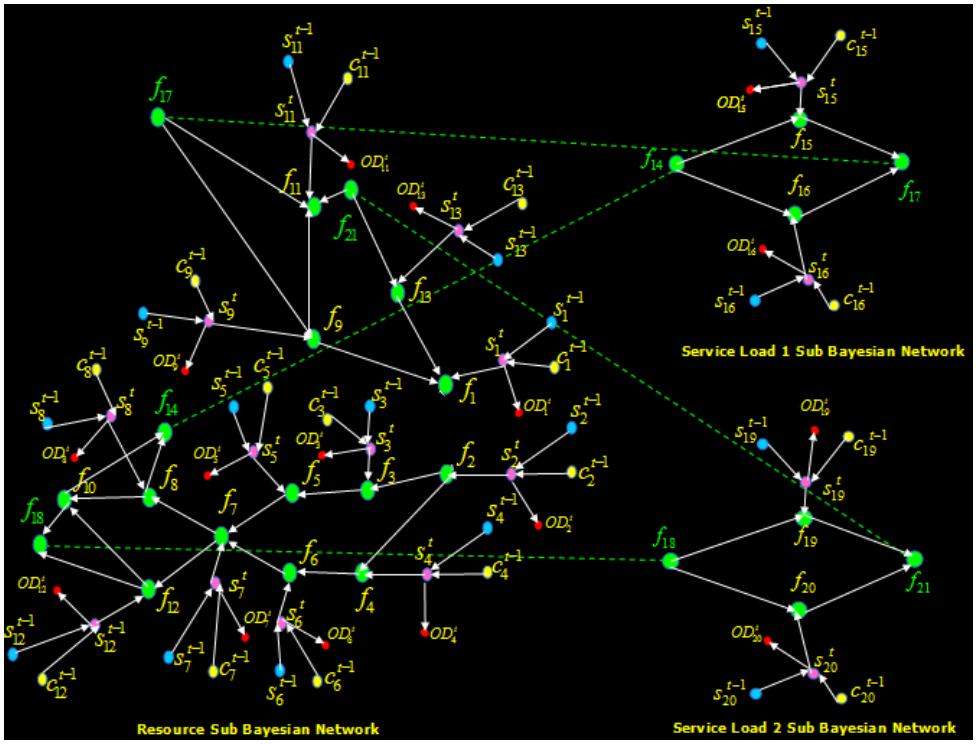


FIGURE 91 MSDBNS OF THE SIMPLIFIED CHILLED WATER SYSTEM

6.2.4.6 Step 6: Insert MSDBNs into Control Structure

In previous steps, the control agents and MSDBNs have been established. The current step will insert the MSDBNs into the control structure to make them work interactively.

Each sub Bayesian network in the MSDBNs will get flow state information from localized sensors/flow meters and commands of components from localized component control agents. The sensors are also represented as agents. Through a complete cycle of CommunicateBelief (CollectBelief followed by DistributeBelief), each sub Bayesian network can make its inference about its local component states, which will be globally consistent based on all available information for the whole system. As mentioned before, each sub Bayesian network is an agent and it provides all basic agent functions as does a control agent. Those sub Bayesian network agents could be situated in different platforms from their corresponding subsystem control agents or in the same platforms. However, in order to reduce communication delays and traffic, it is better to put the sub Bayesian network agent together with its corresponding subsystem control agents, since message passing among them will be heavy. In addition, compatibility and interface matching between Bayesian network agents and control agents will be handled the same way as that for any two control agents described in previous steps.

6.2.4.7 Step 7: Verify and Validate the Designed Control System

Now, the whole control system has been established. It is time to verify, validate and iterate the designed control system. As mentioned in the beginning of this chapter, using a real physical system to test the control system is beyond the scope of this research and therefore, a Flowmaster simulation model is used as a test platform.

Agents are established in Jade, which is completely implemented in Java language, while Flowmaster is thermo-fluid simulation software. The task is to make the Flowmaster model accept control commands from the control agents and the control agents accept information from the simulation results of the Flowmaster model. Although establishing interactions between a Flowmaster model and JADE agents is a time-consuming task, they can be connected through some immediate tools.

Flowmaster models support COM objects named Controllers and Gauges. The Controllers accept information from Visual Basic programs and the Gauges send simulation results to Visual Basic programs. For example, through a Controller, a valve module in Flowmaster model can accept a number between 0 and 1 from an external Visual Basic program to adjust the valve open degree during run time (before/after one iteration or one time-step). Similarly, a Gauge evaluating the flow rate at one point in Flowmaster can send a flow rate value to an external Visual Basic program during run time (before/after one iteration or one time-step).

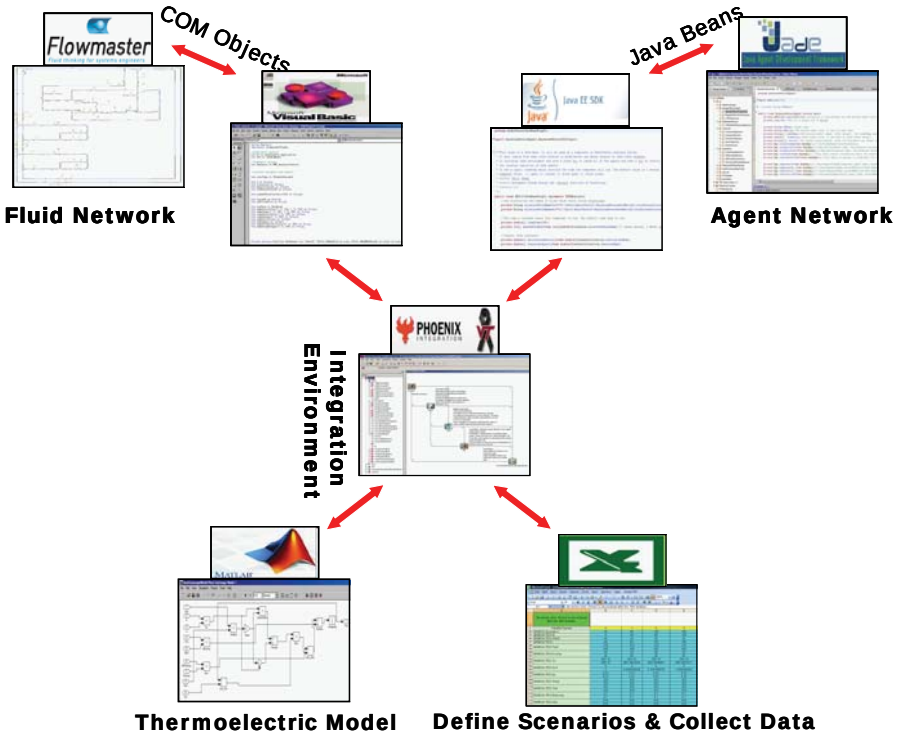


FIGURE 92 INTERACTIONS OF FLOWMASTER MODEL AND AGENTS IN JADE

JADE is open source software fully implemented in Java language and it is very flexible and supports Java Beans. In order to make things convenient, all of the JADE agents are established in the same platform. Since each agent in JADE has its own thread, communication delays between agents can be mimicked by "sleep" of a tread. However,

in these preliminary results, a perfect communication system is assumed (no communication noise, and no communication delays).

Fortunately, ModelCenter of Phoenix Integration as an integration tool that supports both Visual Basic Plug-Ins and Java Bean Plug-Ins. Detailed discussions about Visual Basic Plug-Ins and Java Bean Plug-Ins are provided in Appendix C and Appendix D respectively.

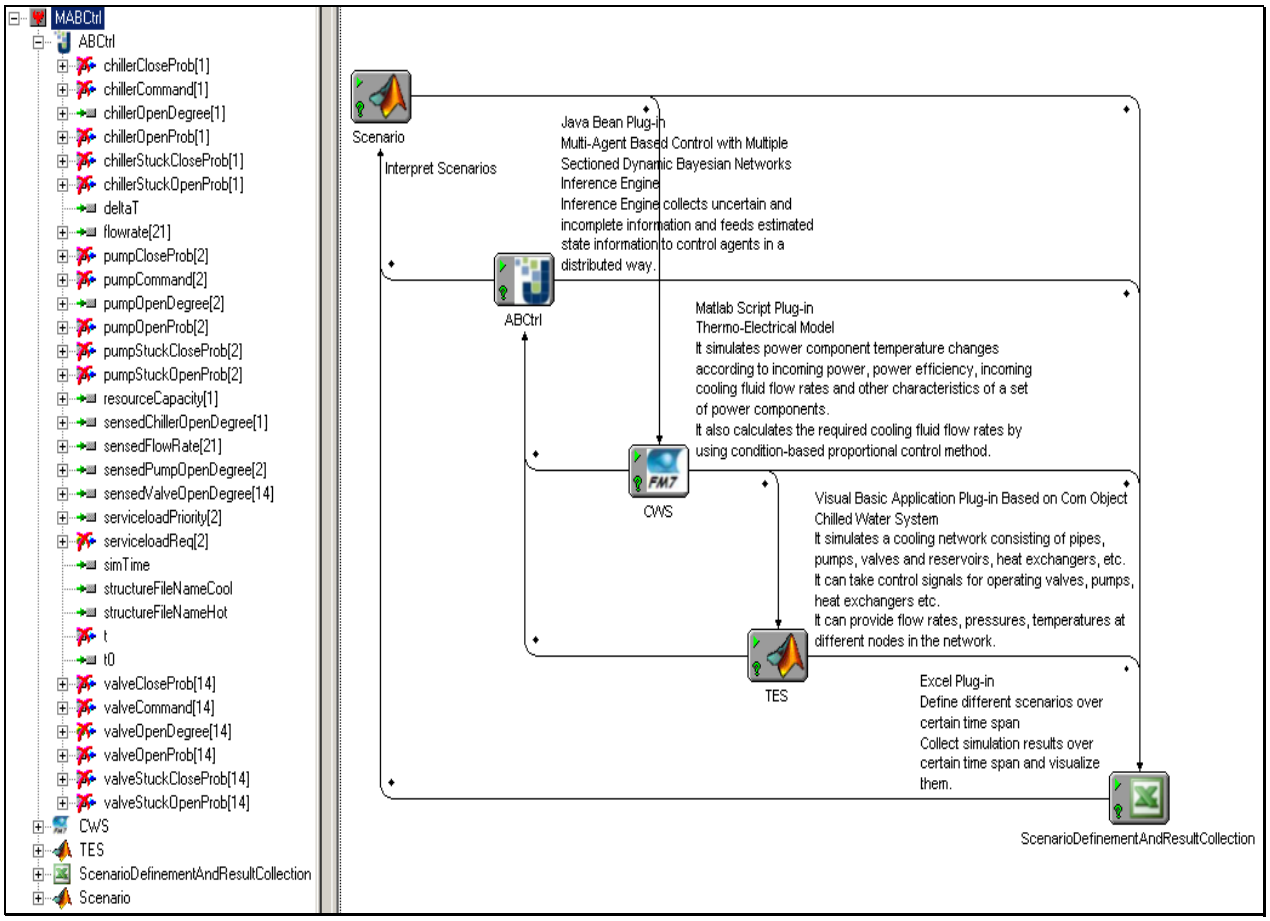


FIGURE 93 THE ENTIRE TEST MODEL IN MODELCENTER ANALYSIS VIEW

In summary, the Flowmaster model and agents in JADE are interacting with each other through Visual Basic and Java programs in ModelCenter as shown in Figure 92. In Figure 92, a third model named TES is also integrated into the ModelCenter model. It is a

simple Simulink model simulating a thermo-electric system, which calculates the chilled water resource requirement for each service load according to the control algorithm discussed in Section 6.2.3. Those service load resource requirements will be inputs for the control system. The entire test environment and models are shown in Figure 93.

6.2.5 Results and Discussions

The integrated model for the application described in detail previously is ready to run. A script scheduler written in VBScript controls the running mode, such as when and how to run each Contribution Analysis (CA), when and how to exchange information between different CAs, how many time steps a CA should run, what outputs should be collected and stored, etc. A detailed description and VBScript code for running the scheduler of the integrated model is included in Appendix F.

TABLE 8 PARAMETERS OF POWER COMPONENT

Terms	Description	Unit
P_{comp}	50/100	kw
η_{comp}	*0.7*	N/A
T_{ci}	*283.15*	K
m_{hx}	616	kg
c_{hx}	400	$J/kg/K$
c_{cf}	4183	$J/kg/K$
A	33	m^2
h	4800	$W/m^2/K$
T_{ctrl}	323 for the Simplified CWS 300.15 for the Notional Ship	K
$T_{co\_threshold}$	353	K
T_{hx_ul}	393	K

The parameters of service loads for power components for differenent scenarios are the same as those listed in Table 8 except for the initial temperature of each service load, which will be reset for each scenario. All of the scenarios are defined in different Excel worksheets. All of the prior distributions and conditional distributions for the Bayesian

network are listed in Appendix G. All of the outputs from every CA for each scenario are collected and stored into a corresponding Excel worksheet. The results shown in Table 9 for each scenario are monitored and plotted, and the figures are presented in the following section. Five different scenarios are listed in Table 10 and discussed in detail in the following sections.

TABLE 9 LIST OF MONITORED AND VISUALIZED RESULTS FOR THE SIMPLIFIED CWS

Pump1 State Distribution vs. Time	**Pump1 Command vs. Time**	**Pump2 State Distribution vs. Time**
Pump2 Command vs. Time	**Valve1 State Distribution vs. Time**	**Valve1 Command vs. Time**
Valve2 State Distribution vs. Time	**Valve2 Command vs. Time**	**Valve7 State Distribution vs. Time**
Valve7 Command vs. Time	**Valve8 State Distribution vs. Time**	**Valve8 Command vs. Time**
Valve11 State Distribution vs. Time	**Valve11 Command vs. Time**	**Valve12 State Distribution vs. Time**
Valve12 Command vs. Time	**Service Load 1 Temperature vs. Time**	**Service Load 1 Flow Rate vs. Time**
Service Load 1 Flow Rate vs. Time	**Service Load 1 Temperature vs. Time**	

TABLE 10 LIST OF 5 DIFFERENT SCENARIOS FOR THE SIMPLIFIED CWS

Scenario / Item	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5
Component State	all undamaged	all undamaged	all undamaged	valve7 STUCKCLOSE at t=440sec, valve11 STUCKCLOSE at t=840sec	valve7 STUCKCLOSE at t=440sec, valve11 STUCKCLOSE at t=840sec
Flow Rate Observability	all observable	all observable	all not observable	all not observable	only f15, f16, f19, f20 observable
Valve Open Degree Observability	all observable	all observable	all not observable	all observable	only valve1, valve2, valve7 observable
Resource Capability (kg/sec)	0.8	0.8	0.8	0.8	0.8
Service Load Initial Temperature (K)	[317 400]	[340 340]	[340 340]	[317 400]	[450 400]
Service Load Incoming Power (kw)	[50 50]	[100 100]	[100 100]	[50 50]	[50 50]
Simulation Time Step (sec)	40	4	4	40	40
Simulation Time (sec)	[0, 1200]	[0, 120]	[0, 120]	[0, 1200]	[0, 1200]

6.2.5.1 Scenario 1 of the Simplified CWS (Nominal Condition 1):

The conditions of scenario 1 are listed in column 2 of Table 10 and the monitored outputs are shown Figure 94 to Figure 113.

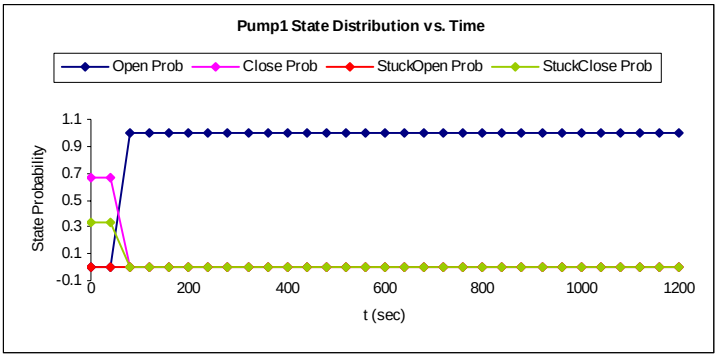


FIGURE 94 SCENARIO 1 (THE SIMPLIFIED CWS): PUMP1 STATE DISTRIBUTION VS. TIME

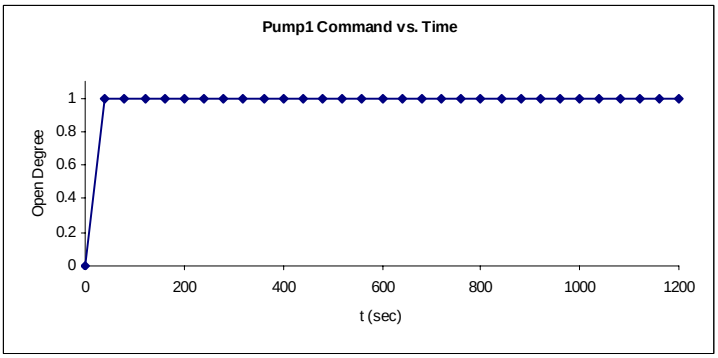


FIGURE 95 SCENARIO 1 (THE SIMPLIFIED CWS): PUMP1 COMMAND VS. TIME

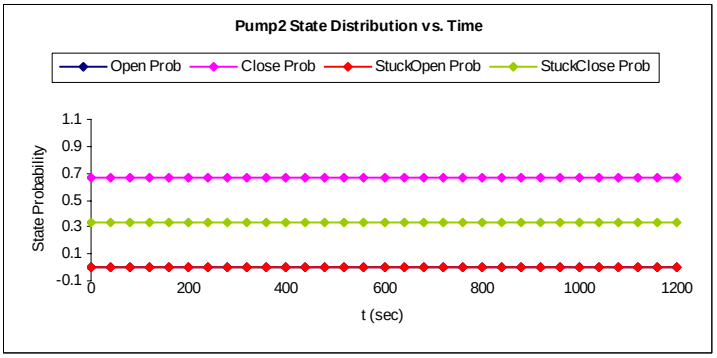


FIGURE 96 SCENARIO 1 (THE SIMPLIFIED CWS): PUMP2 STATE DISTRIBUTION VS. TIME

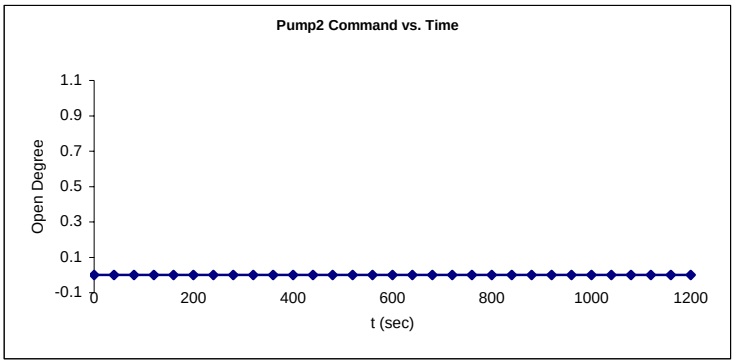


FIGURE 97 SCENARIO 1 (THE SIMPLIFIED CWS): PUMP2 COMMAND VS. TIME

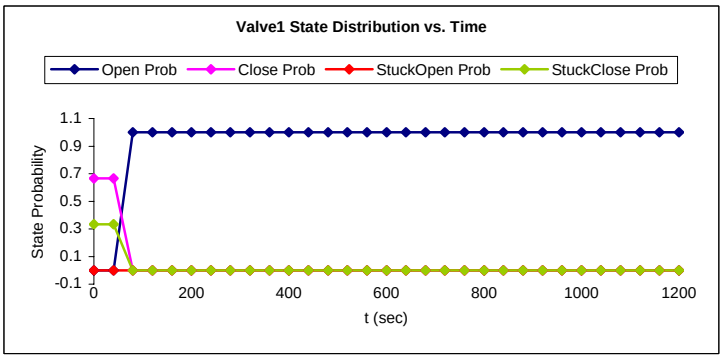


FIGURE 98 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE1 STATE DISTRIBUTION VS. TIME

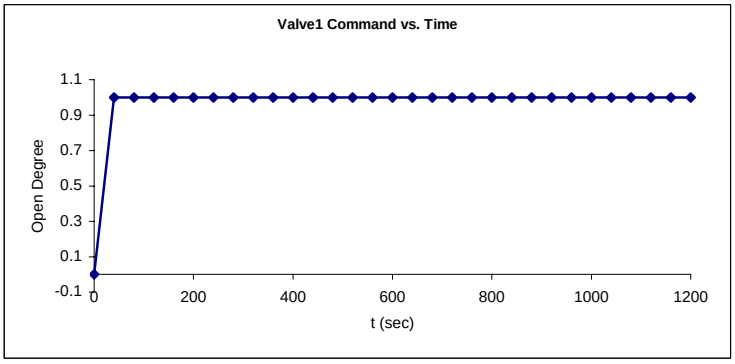


FIGURE 99 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE1 COMMAND VS. TIME

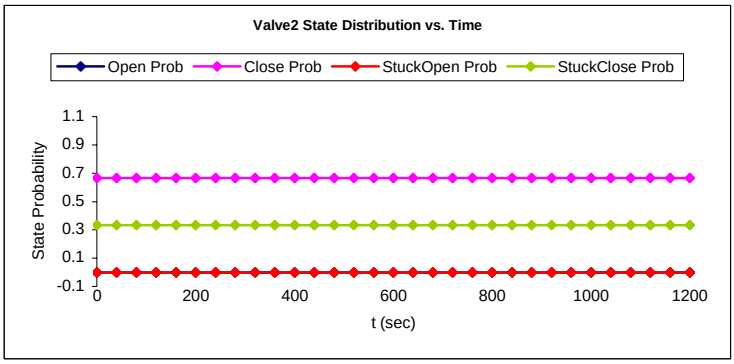


FIGURE 100 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE2 STATE DISTRIBUTION VS. TIME

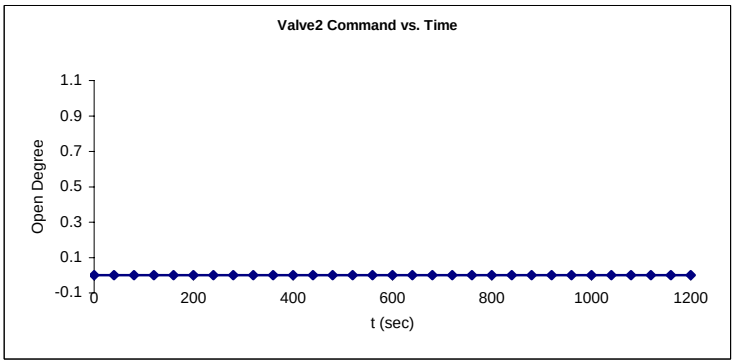


FIGURE 101 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE2 COMMAND VS. TIME

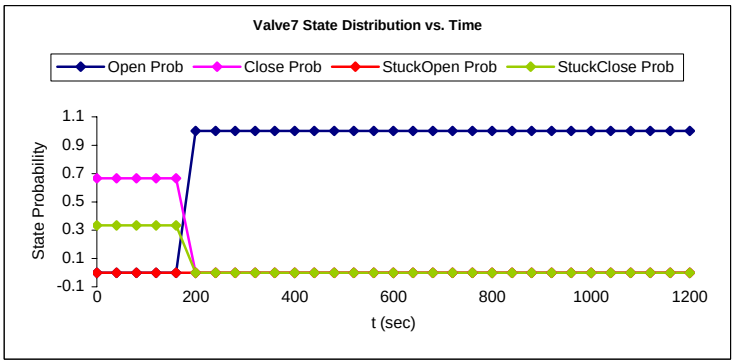


FIGURE 102 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE7 STATE DISTRIBUTION VS. TIME

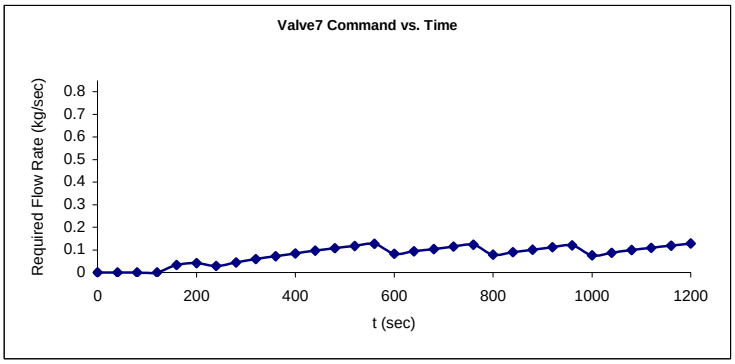


FIGURE 103 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE7 COMMAND VS. TIME

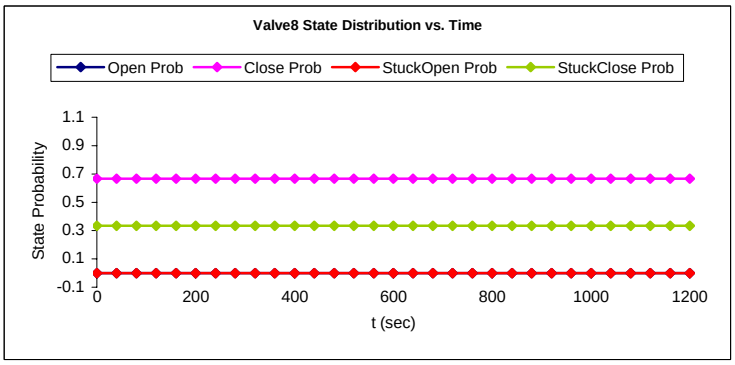


FIGURE 104 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE8 STATE DISTRIBUTION VS. TIME

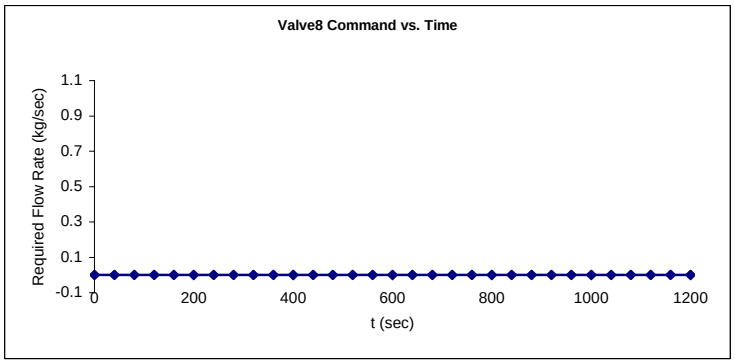


FIGURE 105 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE8 COMMAND VS. TIME

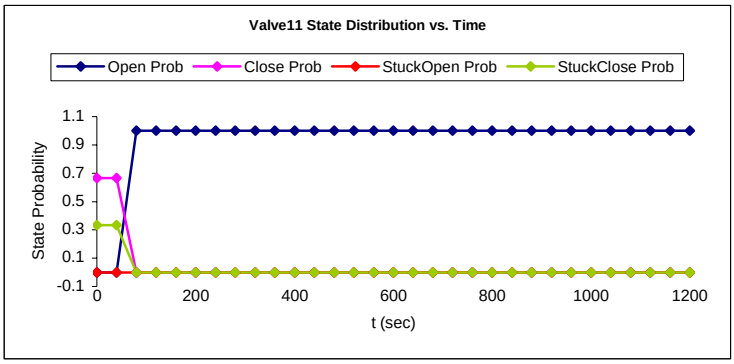


FIGURE 106 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE11 STATE DISTRIBUTION VS. TIME

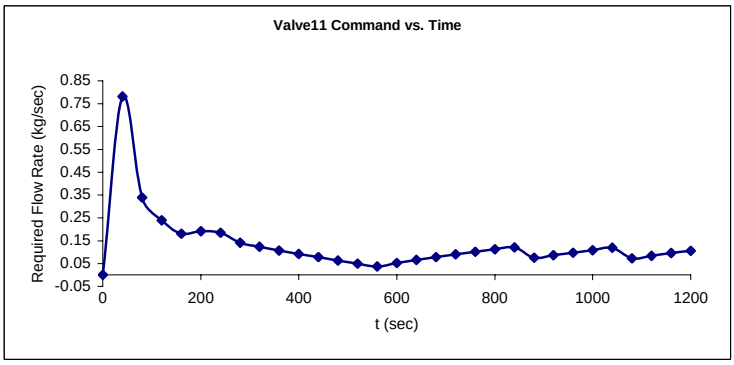


FIGURE 107 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE11 COMMAND VS. TIME

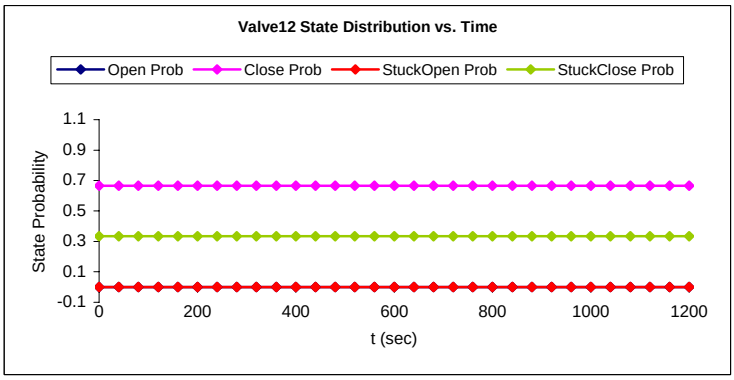


FIGURE 108 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE12 STATE DISTRIBUTION VS. TIME

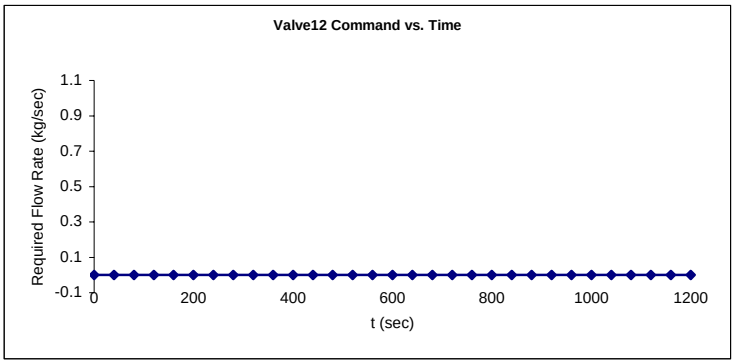


FIGURE 109 SCENARIO 1 (THE SIMPLIFIED CWS): VALVE12 COMMAND VS. TIME

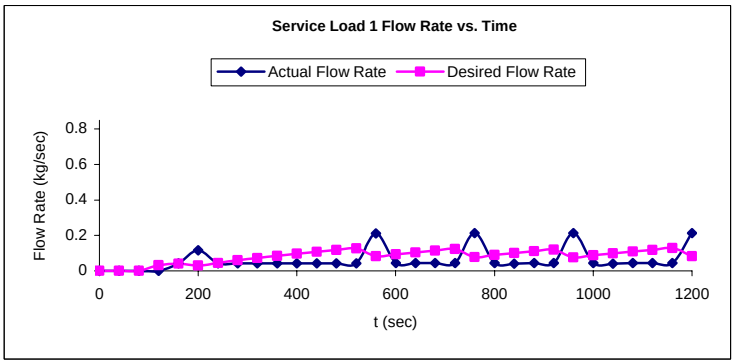


FIGURE 110 SCENARIO 1 (THE SIMPLIFIED CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

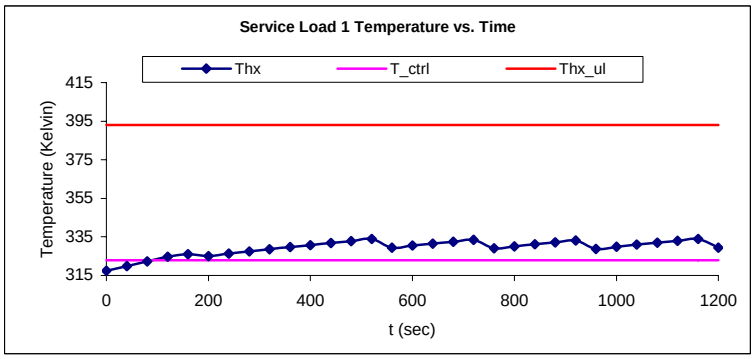


FIGURE 111 SCENARIO 1 (THE SIMPLIFIED CWS): SERVICE LOAD 1 TEMPERATURE VS. TIME

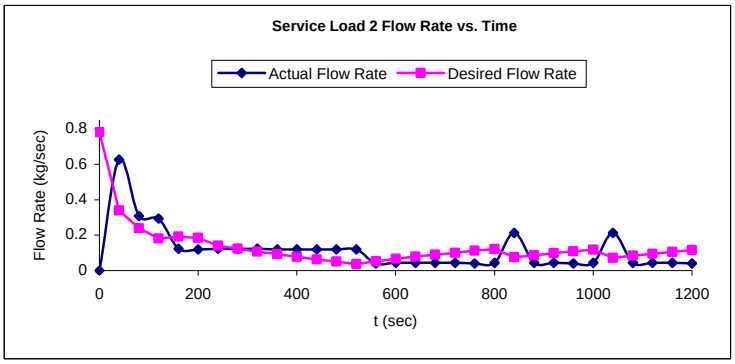


FIGURE 112 SCENARIO 1 (THE SIMPLIFIED CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

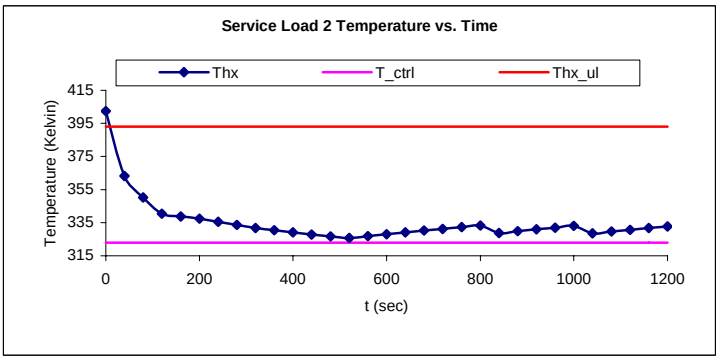


FIGURE 113 SCENARIO 1 (THE SIMPLIFIED CWS): SERVICE LOAD 2 TEMPERATURE VS. TIME

From the initial conditions, at time $t = 0$, the service load 1 temperature is below the control temperature, while the service load 2 temperature exceeds the allowable temperature. Therefore, the service load 2 requires cooling water. The set (pump1 + valve1) is a redundant of the set (pump2 + valve2). Pump1 and valve1 are open to provide cooling water to service load 2 as shown in Figure 94, Figure 95 , Figure 98 and Figure 99. And pump2 and valve2 remain shutdown as shown in Figure 96, Figure 97, Figure 100 and Figure 101. However, checking the state distribution of pump2 and valve2, we found that their states are CLOSE with probability close to 2/3, STUCKCLOSE with probability close to 1/3 and other states with probability close to 0.

The actual states of pump2 and valve2 are CLOSE. These results are reasonable, because the initial state distributions of pump2 and valve2 are uniform distributions. It is impossible to justify a component state STUCKCLOSE from CLOSE based on the observations of CLOSE command and 0 open degree. Similar arguments apply to valve7, valve8 and valve12. Since service load 1 as a power component has 50kw incoming power and the efficiency of the incoming power is 0.7, 30 percent of the incoming power dissipates into service load 1 and causes its temperature to increase. The Thermo-Electrical System (TES) CA calculates the required cooling fluid flow rate according to current service load temperature every 40 seconds. At $t = 120\,\text{sec}$, the temperature of service load 1 is above the control temperature $T_{ctrl} = 323 Kelvin$ as shown in Figure 110; TES CA gets a required flow rate greater than 0 and sends it to the Agent-Based Control (ABCtrl) CA. ABCtrl CA gets the information and tries to redistribute the resource to each service load. From Figure 103, at time $t = 160\,\text{sec}$, the command of valve7 is OPEN, which is delayed for 40 seconds (one iteration of the integration model). Furthermore, the state distribution of valve7 is OPEN with probability close to 1 at time $t = 200\,\text{sec}$. Looking at Figure 110, Figure 111, Figure 112 and Figure 113, we can see that the actual flow rate and required flow rate do not match exactly all the time. This is because the flow rate is controlled by discretly adjusting one valve open degree at a time for one service load and the valve for another service load also affects the flow rate to this service load. In this application, firstly, adjust valve open degree in service load 1 to satisfy the flow rate requirement of service load 1, and then the valve open degree in service load 2 is adjusted to satisfy the flow rate requirement of service load 2, which affects the flow rate in service load 1. That explains why the difference between actual flow rate and desired flow rate in service load 1, as shown in Figure 110 is bigger than that in service load 2, as shown in Figure 112. This reasoning also explains why the temperature of service load 1, shown in Figure 111, fluctuates more than the temperature of service load 2 shown in Figure 113 does. This problem can be solved by adjusting all of the valves

"simultaneously". "Simultaneously" means to reduce the time step for adjusting each valve open degree further. In each small time step, make smaller adjustment sequentially for all of the valves. However, for current situation, the temperatures fluctuate in a tolerable range around the stabilized temperature. In summary, for the nominal case, the control system makes the right decisions and distributes the resource to different service loads accordingly.

6.2.5.2 Scenario 2 of the Simplified CWS (Nominal Condition 2):

The conditions of scenario 2 are listed in column 3 of Table 10 and the monitored outputs are shown from Figure 114 to Figure 117.

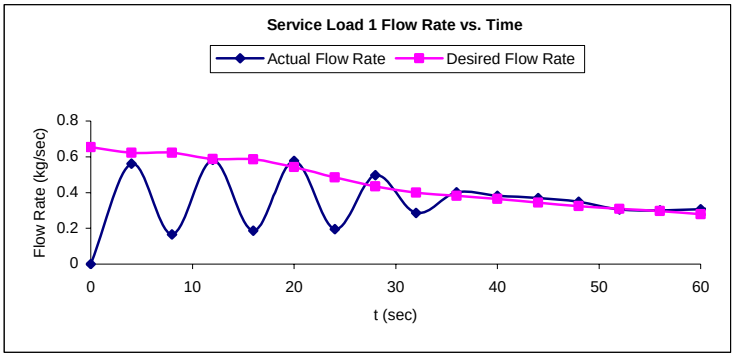


FIGURE 114 SCENARIO 2 (THE SIMPLIFIED CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

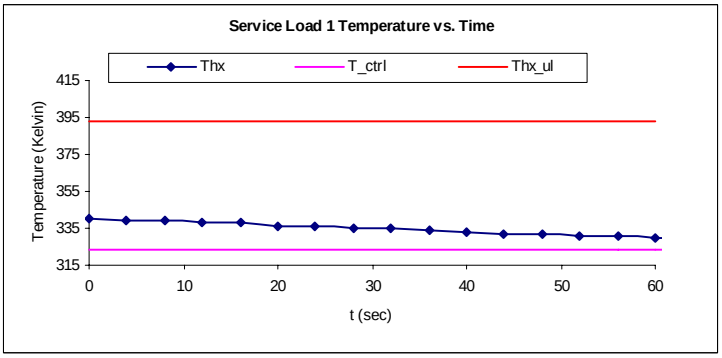


FIGURE 115 SCENARIO 2 (THE SIMPLIFIED CWS): SERVICE LOAD 1 TEMPERATURE VS. TIME

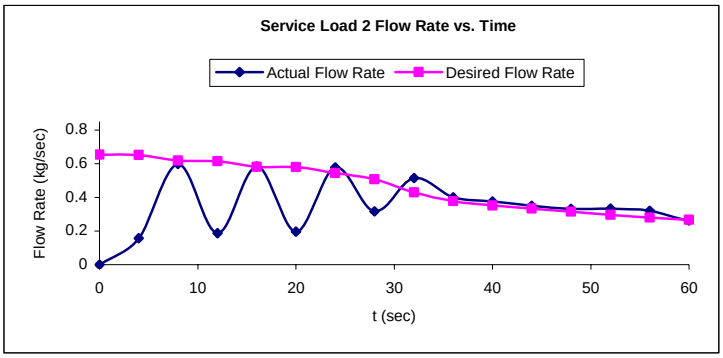


FIGURE 116 SCENARIO 2 (THE SIMPLIFIED CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

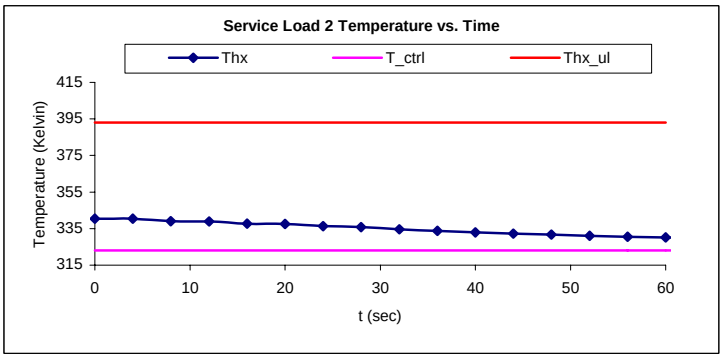


FIGURE 117 SCENARIO 2 (THE SIMPLIFIED CWS): SERVICE LOAD 2 TEMPERATURE VS. TIME

Comparing the results shown in Figure 114 to Figure 117 for scenario 2 with the results shown in Figure 110 to Figure 113 for scenario 1, we can see that with smaller simulation steps and higher control frequecy, the actual flow rate and the actual temperature for each service load change more smoothly. The assumption for the control system processing time in all of the simulations is that the control system can finish the control process in one time step and gives the control commands to the fluid network at the end of each time step. In current simulation environment, all of the agents are created in the same computer and it takes 40 seconds computer time to update the complete set of agents' states.

6.2.5.3 Scenario 3 of the Simplified CWS (No Observations):

The conditions of scenario 3 are listed in column 4 of Table 10 and the monitored outputs are shown from Figure 118 to Figure 137.

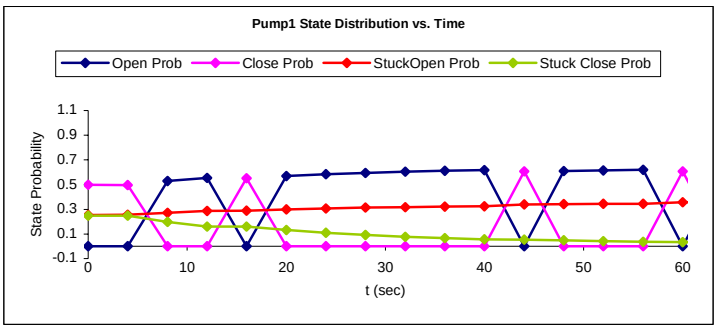


FIGURE 118 SCENARIO 3 (THE SIMPLIFIED CWS): PUMP1 STATE DISTRIBUTION VS. TIME

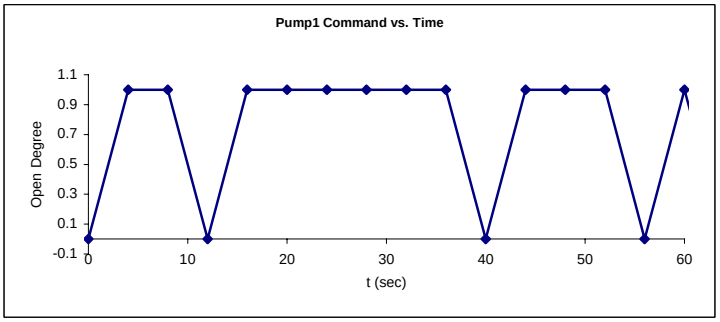


FIGURE 119 SCENARIO 3 (THE SIMPLIFIED CWS): PUMP1 COMMAND VS. TIME

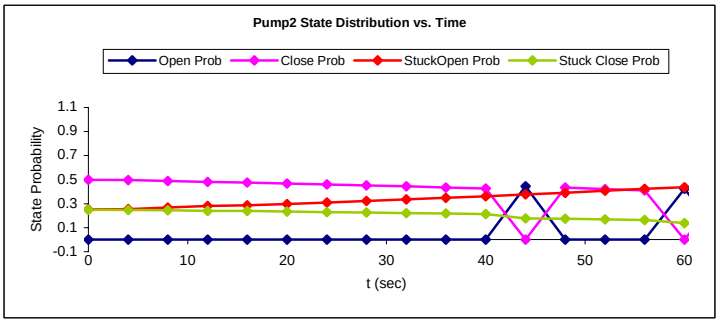


FIGURE 120 SCENARIO 3 (THE SIMPLIFIED CWS): PUMP2 STATE DISTRIBUTION VS. TIME

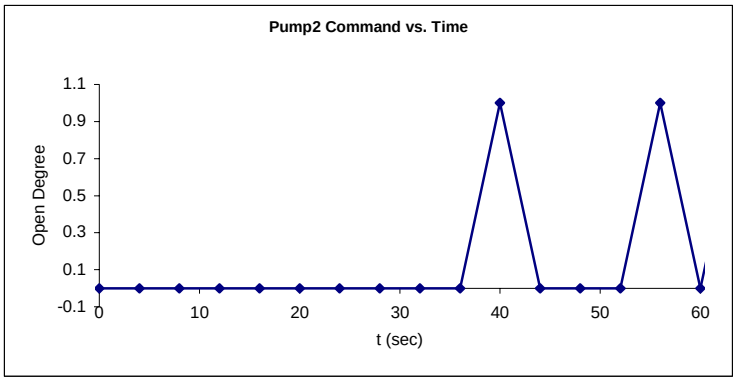


FIGURE 121 SCENARIO 3 (THE SIMPLIFIED CWS): PUMP2 COMMAND VS. TIME

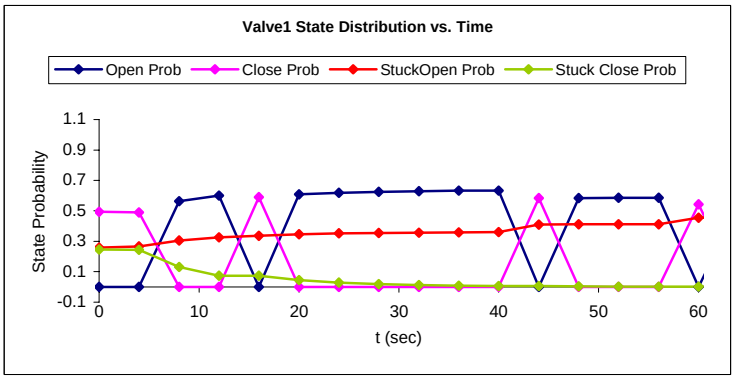


FIGURE 122 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE1 STATE DISTRIBUTION VS. TIME

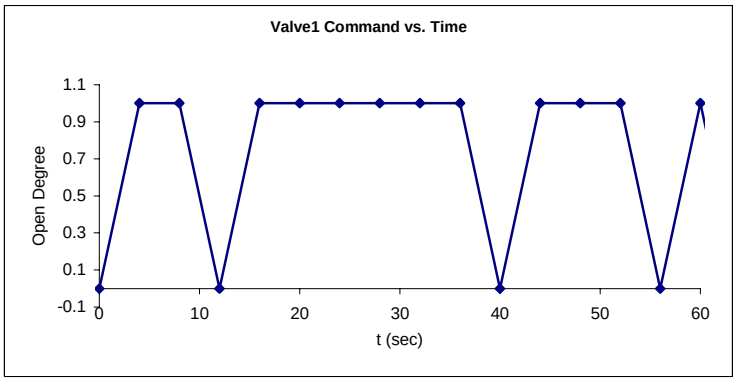


FIGURE 123 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE1 COMMAND VS. TIME

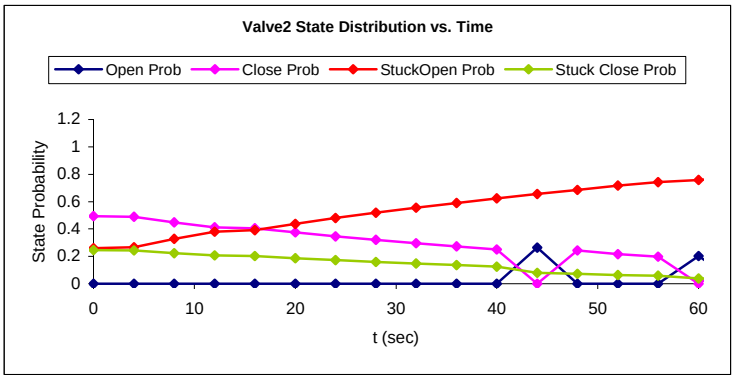


FIGURE 124 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE2 STATE DISTRIBUTION VS. TIME

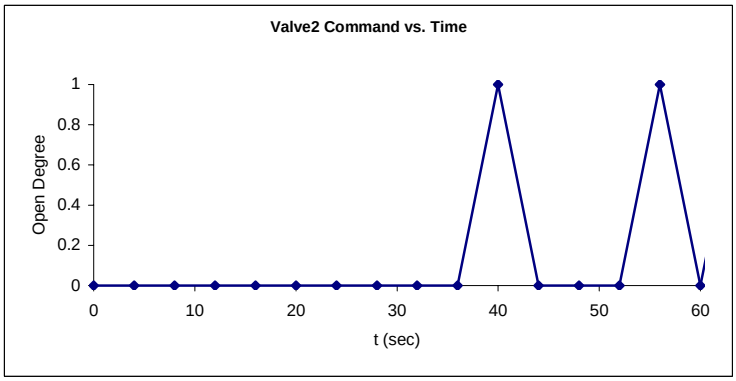


FIGURE 125 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE2 COMMAND VS. TIME

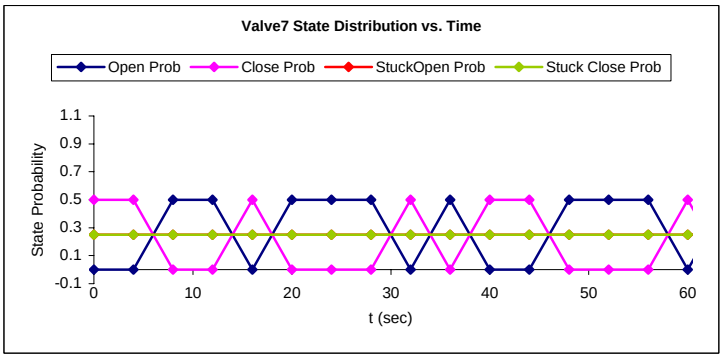


FIGURE 126 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE7 STATE DISTRIBUTION VS. TIME

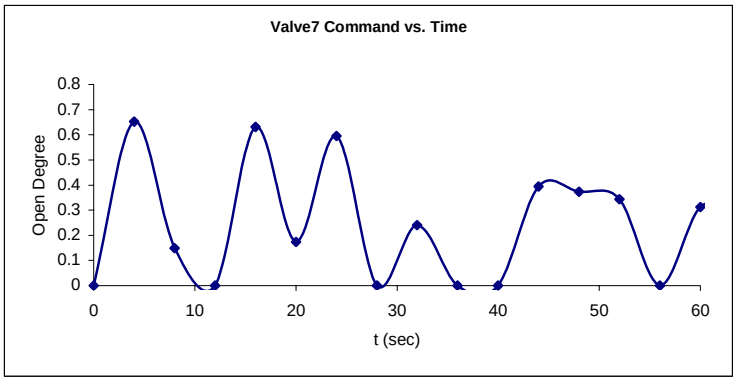


FIGURE 127 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE7 COMMAND VS. TIME

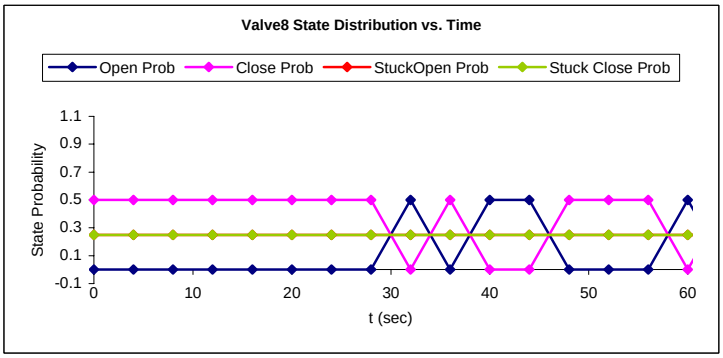


FIGURE 128 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE8 STATE DISTRIBUTION VS. TIME

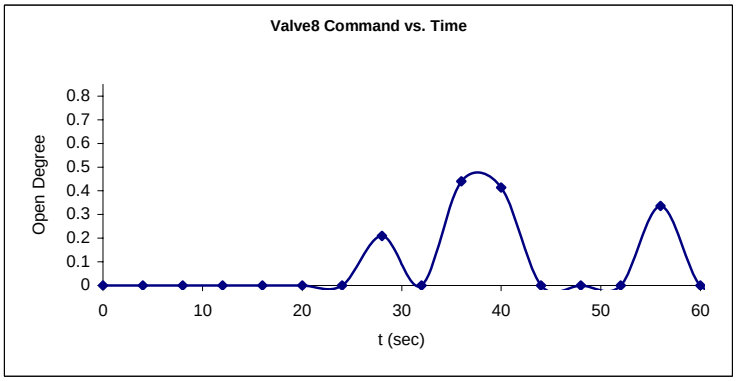


FIGURE 129 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE8 COMMAND VS. TIME

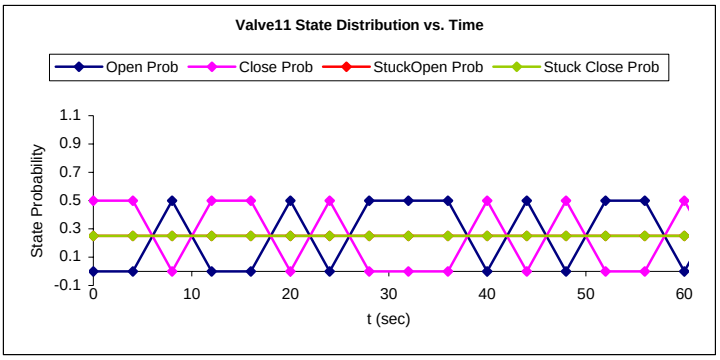


FIGURE 130 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE11 STATE DISTRIBUTION VS. TIME

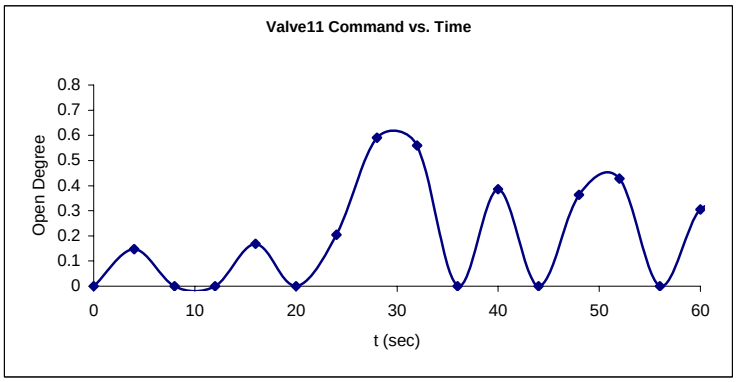


FIGURE 131 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE11 COMMAND VS. TIME

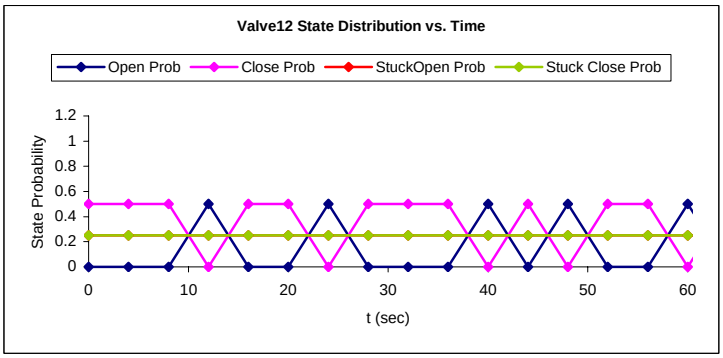


FIGURE 132 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE12 STATE DISTRIBUTION VS. TIME

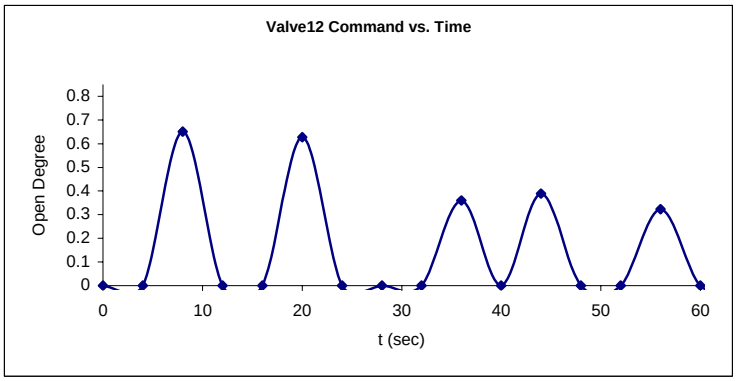


FIGURE 133 SCENARIO 3 (THE SIMPLIFIED CWS): VALVE12 COMMAND VS. TIME

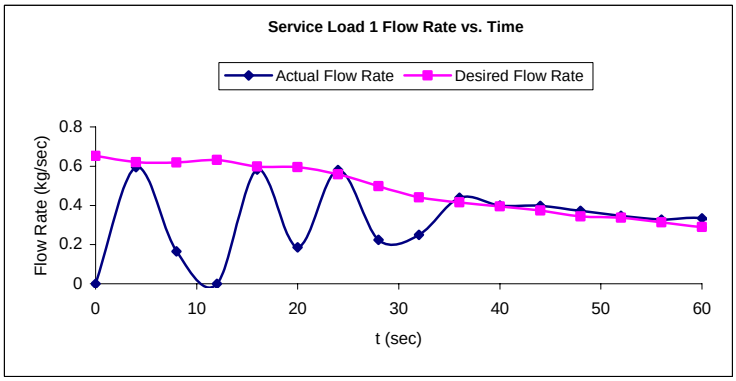


FIGURE 134 SCENARIO 3 (THE SIMPLIFIED CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

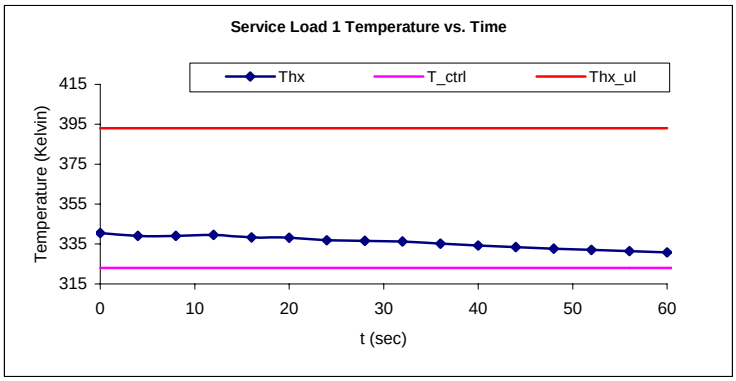


FIGURE 135 SCENARIO 3 (THE SIMPLIFIED CWS): SERVICE LOAD 1 TEMPERATURE VS. TIME

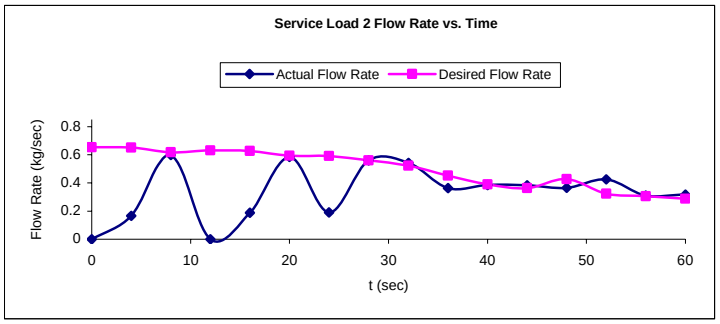


FIGURE 136 SCENARIO 3 (THE SIMPLIFIED CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

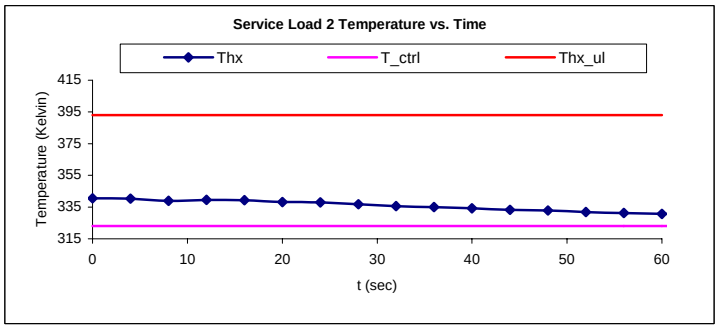


FIGURE 137 SCENARIO 3 (THE SIMPLIFIED CWS): SERVICE LOAD 2 TEMPERATURE VS. TIME

Comparing scenario 3 with scenario 1 (nominal conditions 1), there is no observation in scenario 3 except the flow rate point f_2 for breaking the directed cycle in the Bayesian networks. The prior distribution of each component state is uniform. The only available information in this scenario is the control command to each component at each time step. Based on the command information and the cause-effect relationships interweaved in the established Bayesian networks, the MSDBNs perform inferences and provide the state estimation of each component to the control system. The control system accepts component state distribution information and selects a component state proportional to its state likihood. Since the prior distribution is uniform, initially, each state for a component has the equal chance to be selected, which causes a component state to switch frequently

as the simulation proceeds. For example, checking the valve7 state distribution versus time in Figure 126 and valve7 command versus time in Figure 127, we can see that the valve7 state estimation does not converge to one state and the control system picks up its state proportionally to the state likelihood and the control command to valve7 flips frequently between OPEN and CLOSE. In summary, without observations and only with a uniform prior distribution of each component state, the state estimation for each component from MSDBNs inference engine can not converge to one state, i.e., MSDBNs inference engine can not detect component states.

6.2.5.4 Scenario 4 of the Simplified CWS (Damage Condition 1):

The conditions of scenario 4 are listed in column 5 of Table 10 and the monitored outputs are shown from Figure 138 to Figure 155.

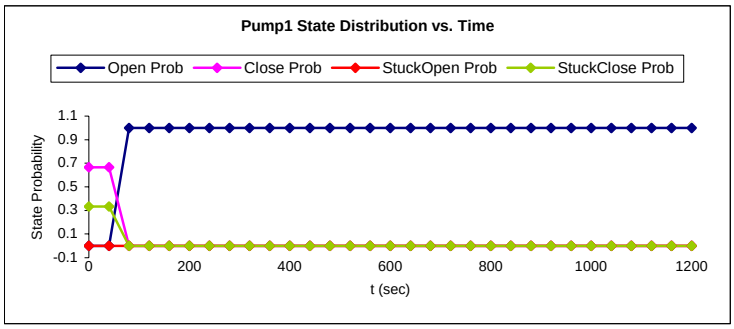


FIGURE 138 SCENARIO 4 (THE SIMPLIFIED CWS): PUMP1 STATE DISTRIBUTION VS. TIME

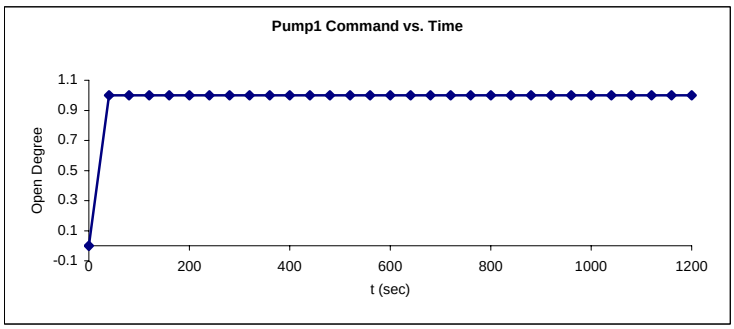


FIGURE 139 SCENARIO 4 (THE SIMPLIFIED CWS): PUMP1 COMMAND VS. TIME

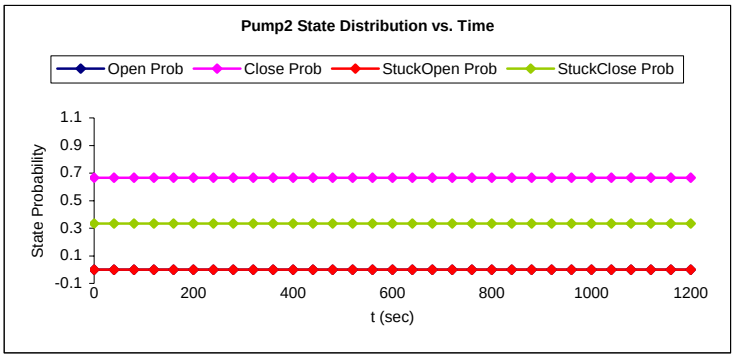


FIGURE 140 SCENARIO 4 (THE SIMPLIFIED CWS): PUMP2 STATE DISTRIBUTION VS. TIME

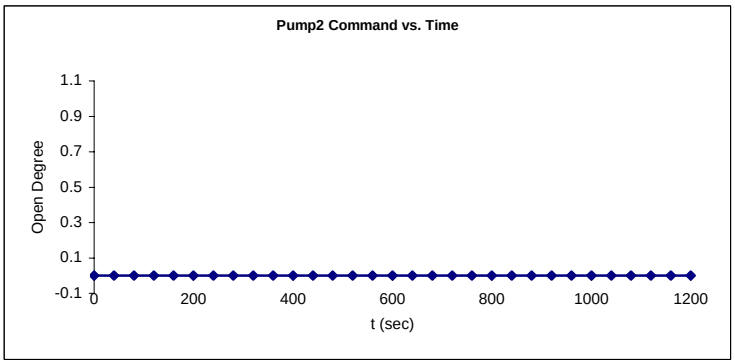


FIGURE 141 SCENARIO 4 (THE SIMPLIFIED CWS): PUMP2 COMMAND VS. TIME

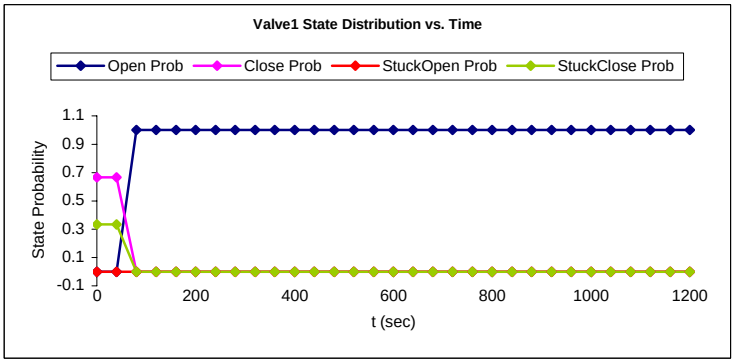


FIGURE 142 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE1 STATE DISTRIBUTION VS. TIME

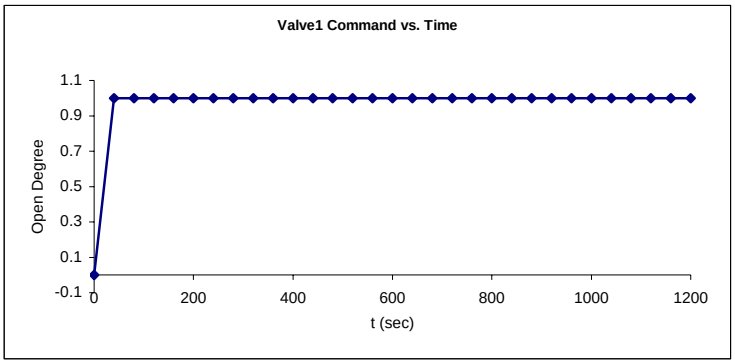


FIGURE 143 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE1 COMMAND VS. TIME

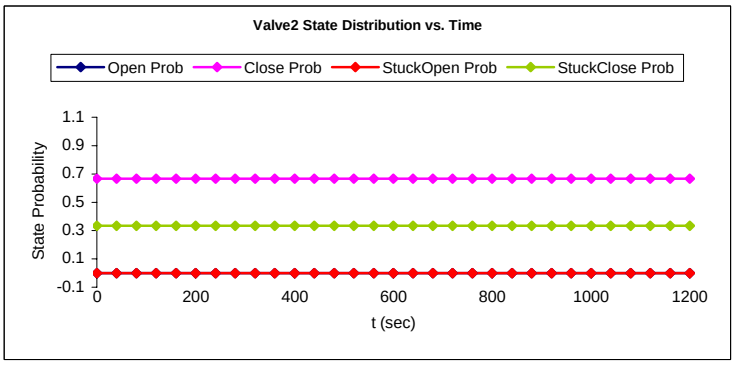


FIGURE 144 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE2 STATE DISTRIBUTION VS. TIME

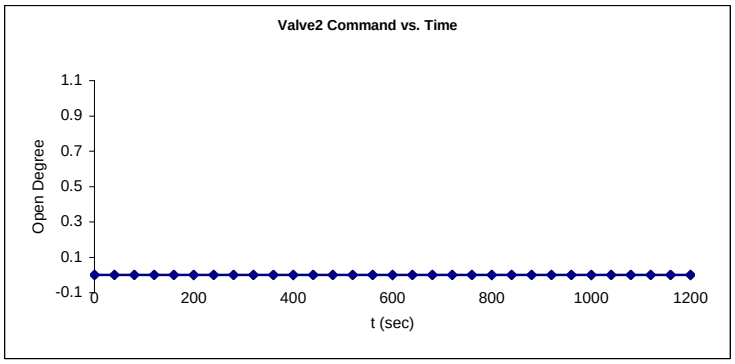


FIGURE 145 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE2 COMMAND VS. TIME

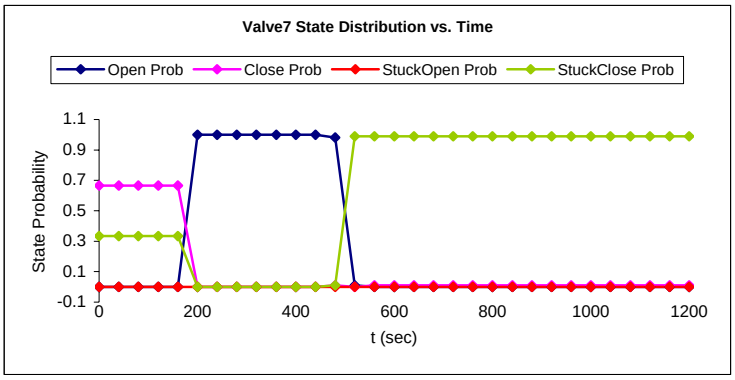


FIGURE 146 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE7 STATE DISTRIBUTION VS. TIME

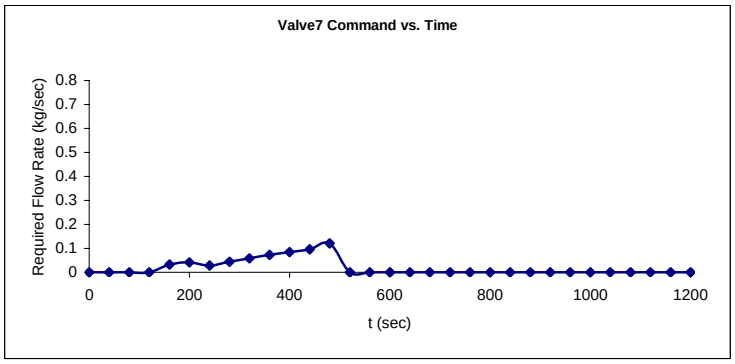


FIGURE 147 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE7 COMMAND VS. TIME

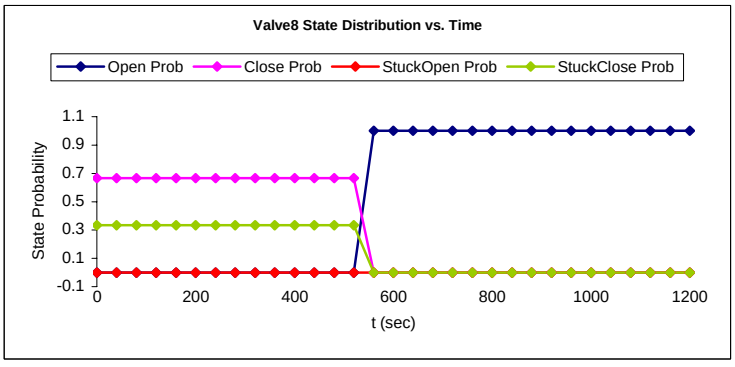


FIGURE 148 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE8 STATE DISTRIBUTION VS. TIME

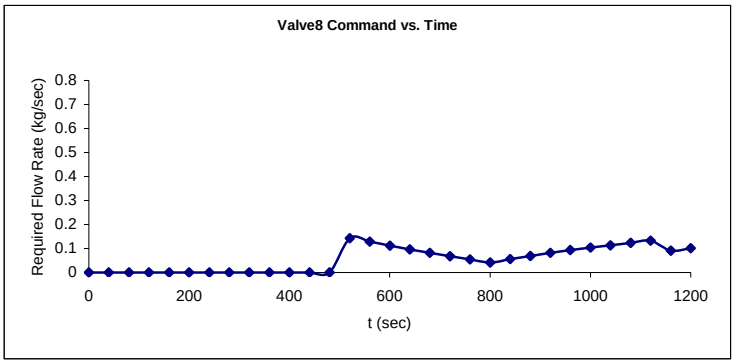


FIGURE 149 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE8 COMMAND VS. TIME

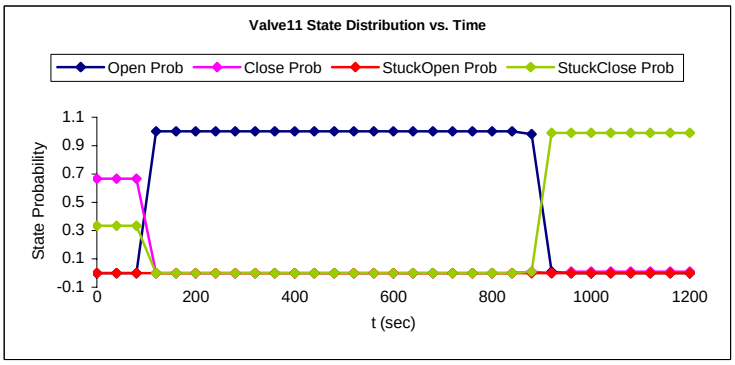


FIGURE 150 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE11 STATE DISTRIBUTION VS. TIME

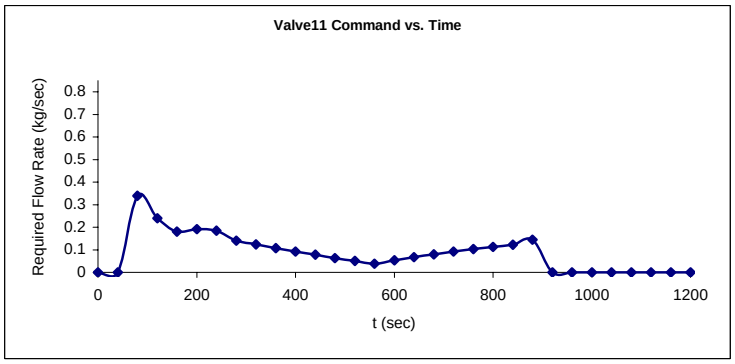


FIGURE 151 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE11 COMMAND VS. TIME

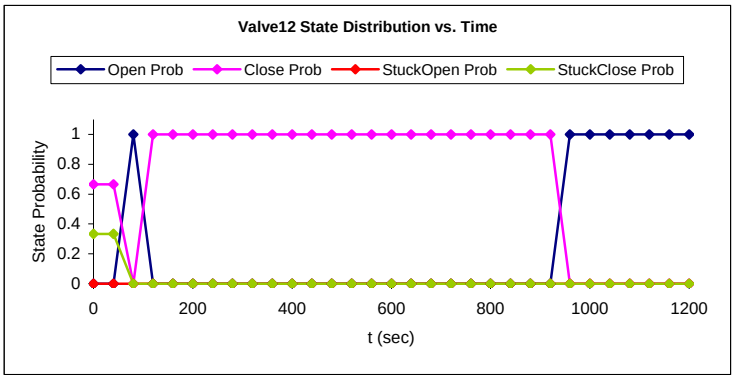


FIGURE 152 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE12 STATE DISTRIBUTION VS. TIME

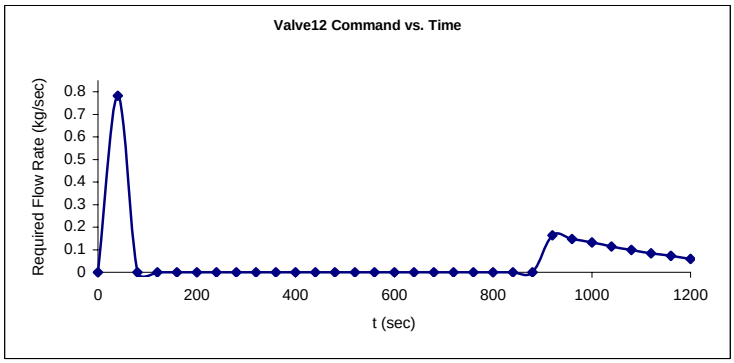


FIGURE 153 SCENARIO 4 (THE SIMPLIFIED CWS): VALVE12 COMMAND VS. TIME

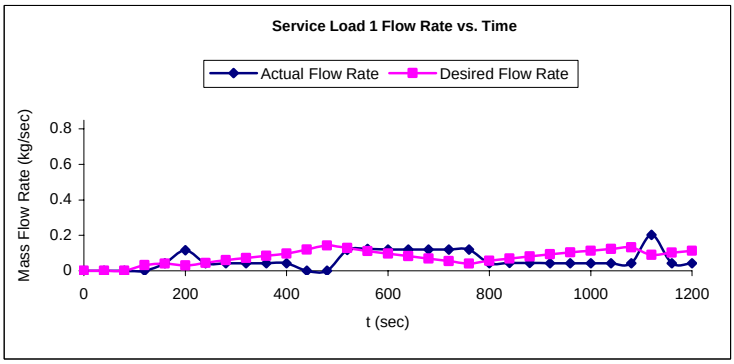


FIGURE 154 SCENARIO 4 (THE SIMPLIFIED CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

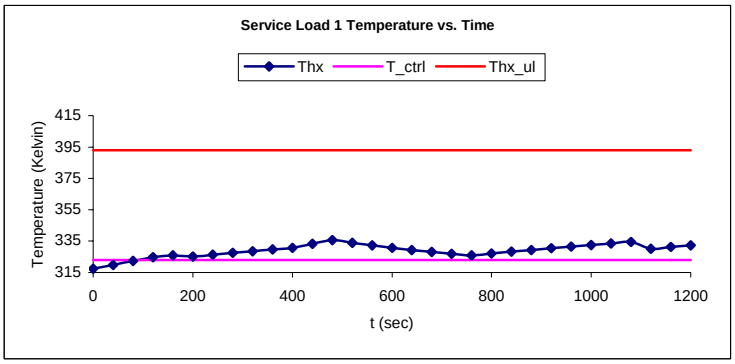


FIGURE 155 SCENARIO 4 (THE SIMPLIFIED CWS): SERVICE LOAD 1 TEMPERATURE VS. TIME

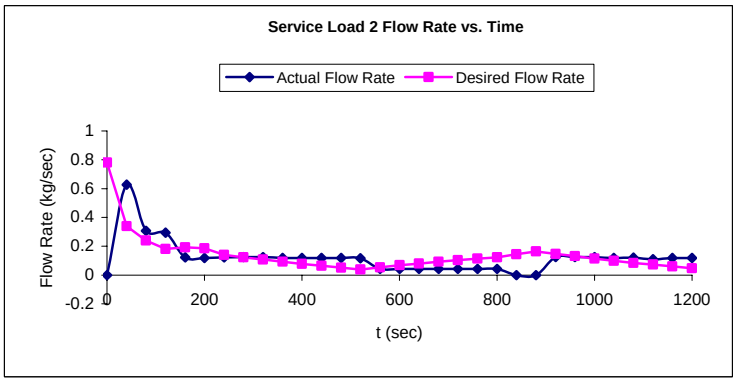


FIGURE 156 SCENARIO 4 (THE SIMPLIFIED CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

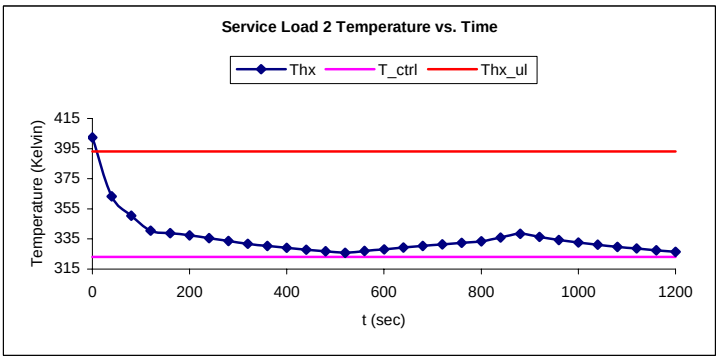


FIGURE 157 SCENARIO 4 (THE SIMPLIFIED CWS): SERVICE LOAD 2 TEMPERATURE VS. TIME

From the initial conditions, before time $t = 440\,\text{sec}$, scenario 4 is as the same as scenario1 except no flow rate is observable. Compare Figure 138 through Figure 155 with Figure 94 through Figure 113, we can see the estimation results and control results in scenario 4 are close to those in scenario 1. At time $t = 440\,\text{sec}$, valve7 becomes STUCKCLOSE. The distributed Baysian inference engine detects the valve7 state change at time $t = 520\,\text{sec}$. At time $t = 440\,\text{sec}$, an OPEN command is given to valve7 from the control system; at time $t = 480\,\text{sec}$, the fluid network gives the valve open degree information back to the control system; through data processing, at time $t = 520\,\text{sec}$, the inference engine embedded in this control system gives that valve7 is STUCKCLOSE with probability close to 1. At the same time, an OPEN command is sent by the control system to valve8 as a backup of valve7 in the fluid network system and valve8 state distribution is changed to OPEN with probability close to 1 at time $t = 560\,\text{sec}$. Similarly, the valve11 is detected being STUCKCLOSE 80 seconds (two iterations) later after its state changes at time $t = 840\,\text{sec}$ by the distributed inference engine; an OPEN command to valve12 is initiated by the control system at time $t = 920\,\text{sec}$ and valve12 state distribution is changed into OPEN with probability close to 1 at time $t = 960\,\text{sec}$. Those detections of state changes for valve7, valve8, valve11 and valve12 are shown in Figure 146, Figure 148, Figure 150 and Figure 152 respectively. Due to the delays of two iterations (80 seconds) for the state change detections, there is no flow through service load 1 during time $t = 440\,\text{sec}$ to time $t = 520\,\text{sec}$, as seen in Figure 154. Similarly, there is no flow through service load 2 during time $t = 840\,\text{sec}$ to time $t = 920\,\text{sec}$, as seen in Figure 156. The delays are also reflected in steeper temperature gradients in service load 1 and service load 2 as shown in Figure 155 and Figure 157 respectively. In summary, without any flow rate observation and only with component open degree observations, the inference engine can detect component damages and the control system can reconfigure the entire system by switching from damaged components to their corresponding redundant ones to redistribute system resource accordingly.

6.2.5.5 Scenario 5 of the Simplified CWS (Damage Condition 2):

The conditions of scenario 5 are listed in column 6 of Table 10 and the monitored outputs are shown from Figure 158 to Figure 177.

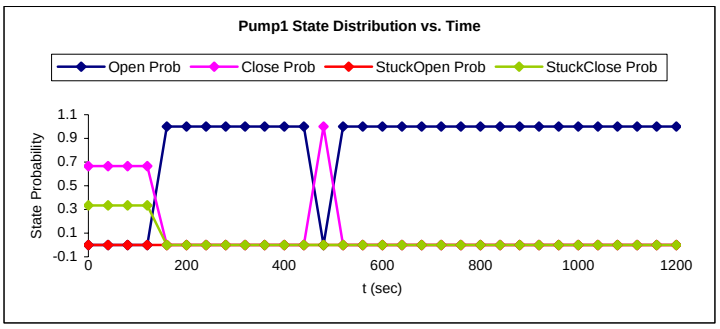


FIGURE 158 SCENARIO 5 (THE SIMPLIFIED CWS): PUMP1 STATE DISTRIBUTION VS. TIME

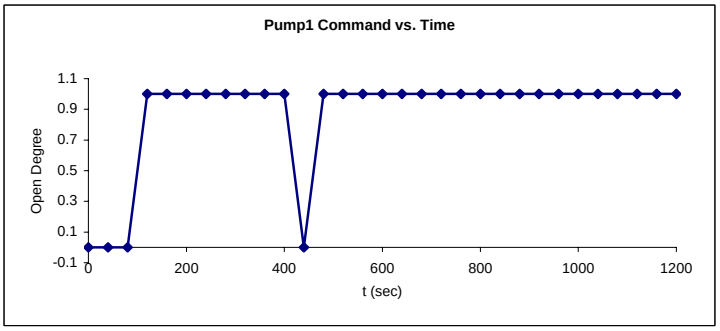


FIGURE 159 SCENARIO 5 (THE SIMPLIFIED CWS): PUMP1 COMMAND VS. TIME

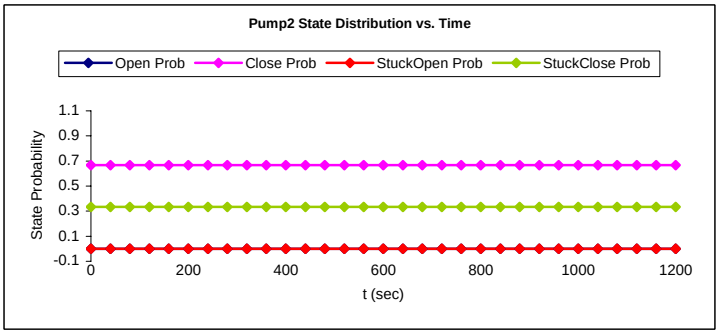


FIGURE 160 SCENARIO 5 (THE SIMPLIFIED CWS): PUMP2 STATE DISTRIBUTION VS. TIME

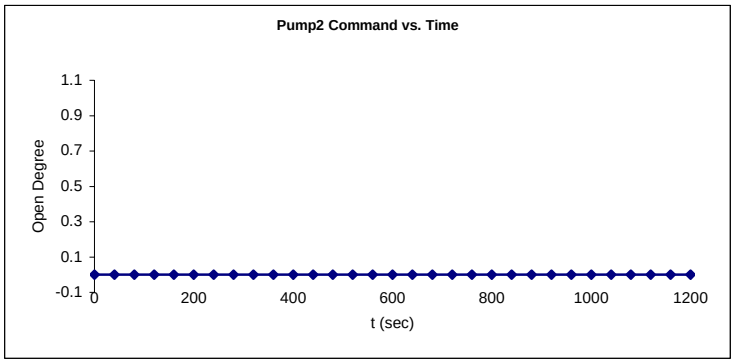


FIGURE 161 SCENARIO 5 (THE SIMPLIFIED CWS): PUMP2 COMMAND VS. TIME

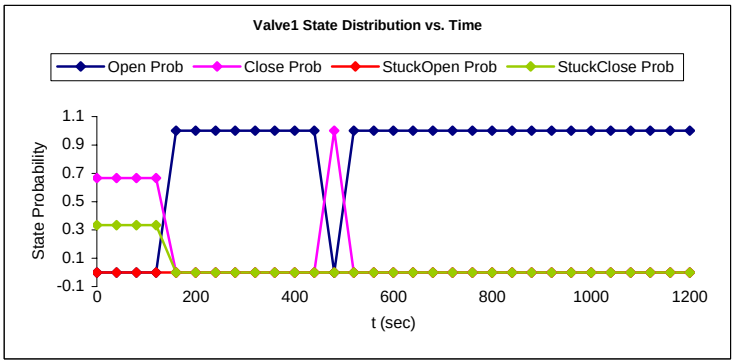


FIGURE 162 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE1 STATE DISTRIBUTION VS. TIME

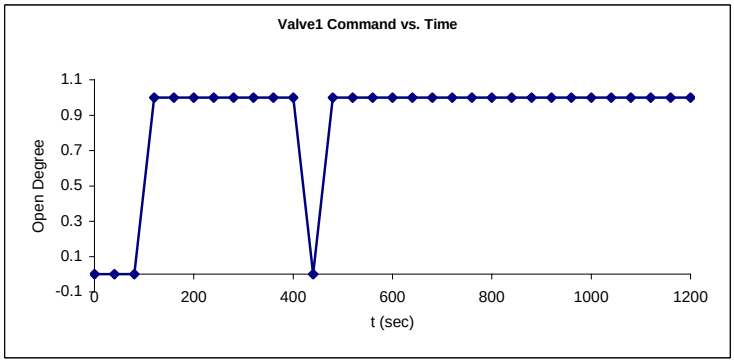


FIGURE 163 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE1 COMMAND VS. TIME

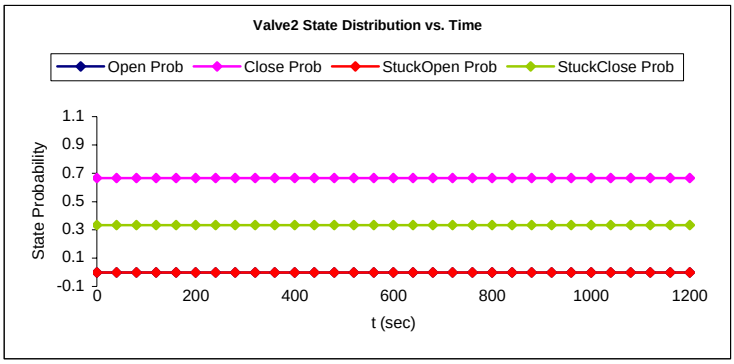


FIGURE 164 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE2 STATE DISTRIBUTION VS. TIME

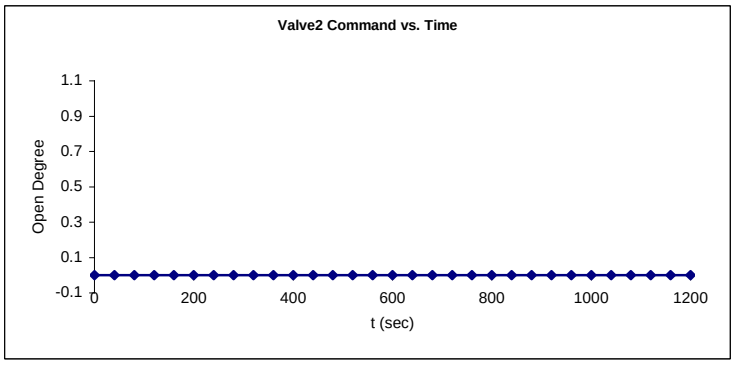


FIGURE 165 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE2 COMMAND VS. TIME

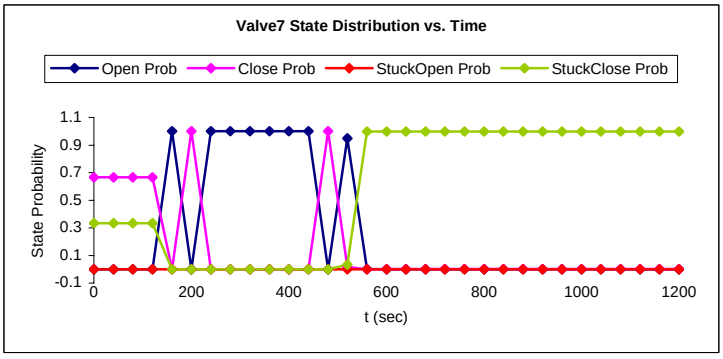


FIGURE 166 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE7 STATE DISTRIBUTION VS. TIME

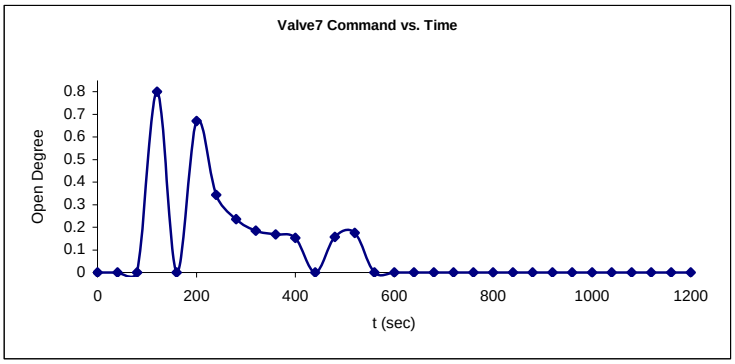


FIGURE 167 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE7 COMMAND VS. TIME

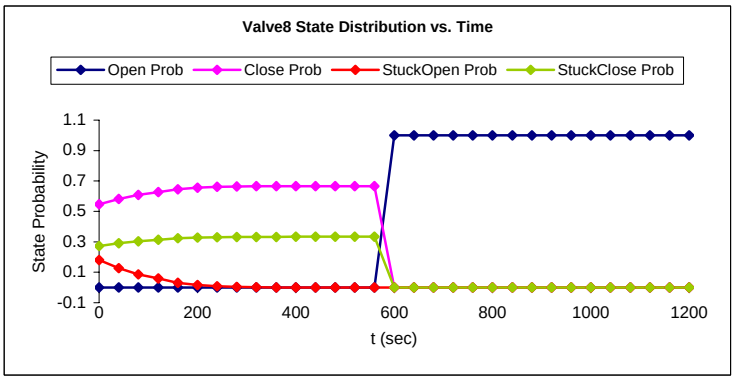


FIGURE 168 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE8 STATE DISTRIBUTION VS. TIME

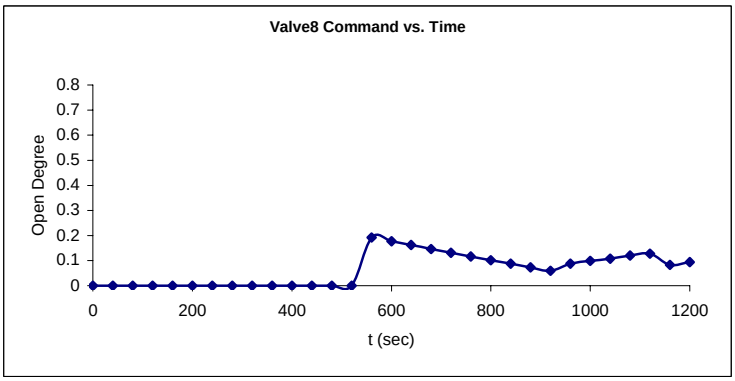


FIGURE 169 SCENARIO 5 (THE SIMPLIFIED CWS): VAVLE8 COMMAND VS. TIME

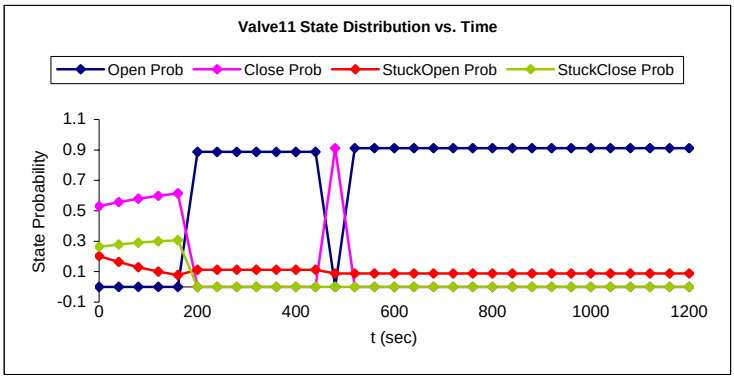


FIGURE 170 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE11 STATE DISTRIBUTION VS. TIME

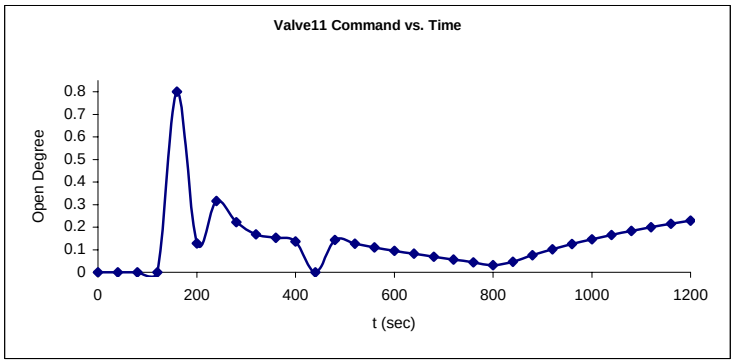


FIGURE 171 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE11 COMMAND VS. TIME

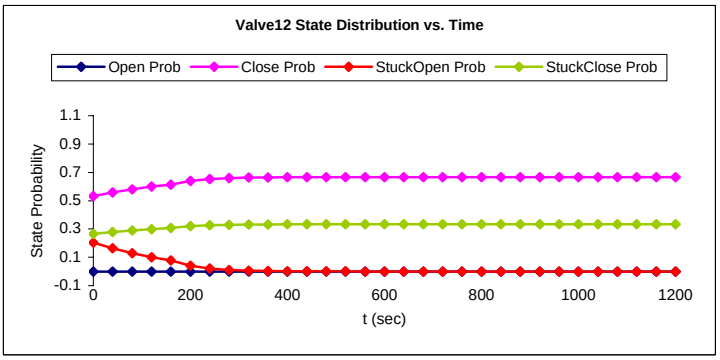


FIGURE 172 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE12 STATE DISTRIBUTION VS. TIME

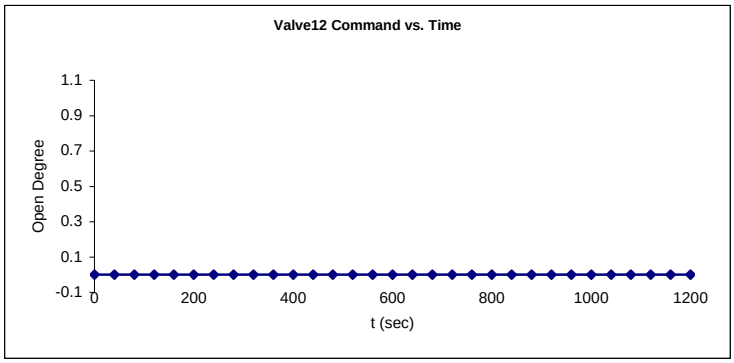


FIGURE 173 SCENARIO 5 (THE SIMPLIFIED CWS): VALVE12 COMMAND VS. TIME

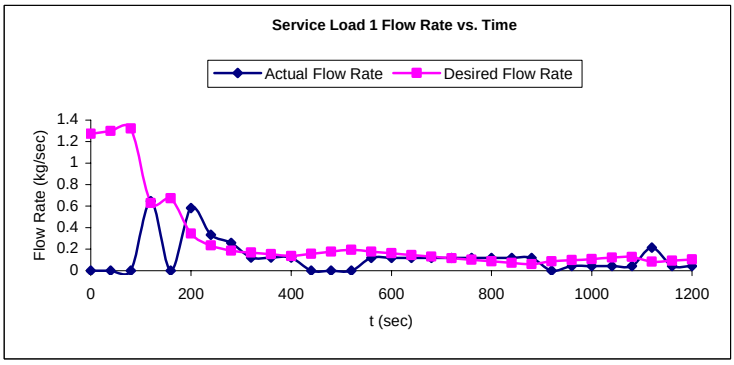


FIGURE 174 SCENARIO 5 (THE SIMPLIFIED CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

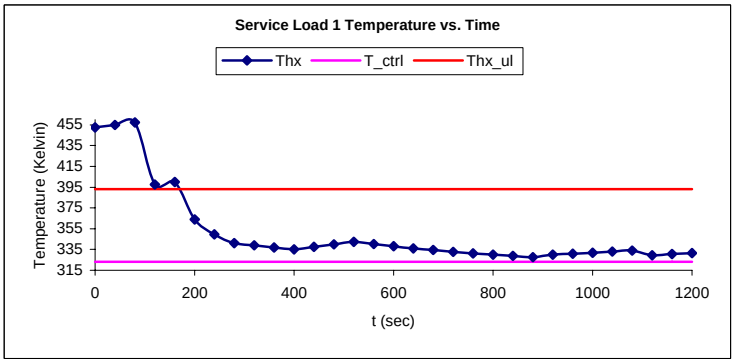


FIGURE 175 SCENARIO 5 (THE SIMPLIFIED CWS): SERVICE LOAD 1 TEMPERATURE VS. TIME

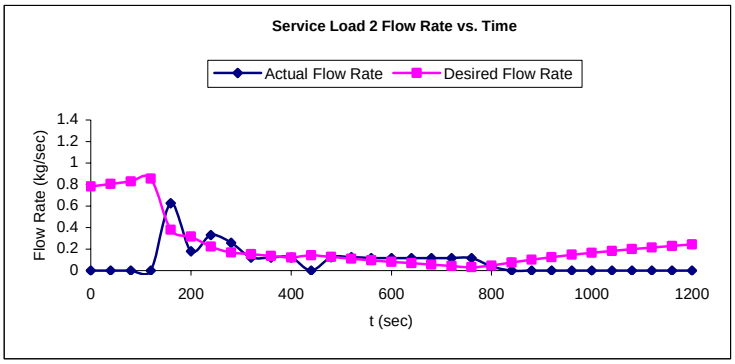


FIGURE 176 SCENARIO 5 (THE SIMPLIFIED CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

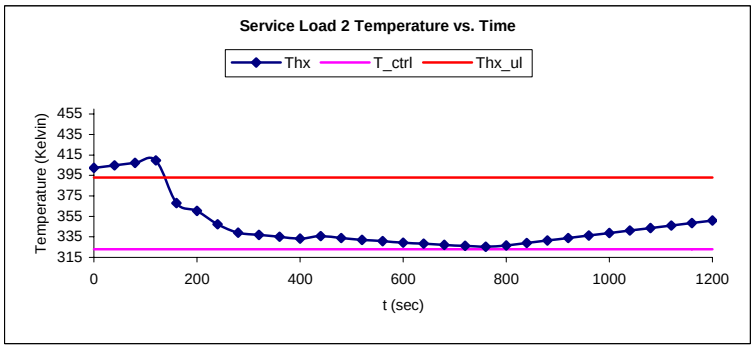


FIGURE 177 SCENARIO 5 (THE SIMPLIFIED CWS): SERVICE LOAD 2 TEMPERATURE VS. TIME

From the initial conditions, we can see that both of service load 1 and service load 2's temperatures far exceed the control temperature of 323 Kelvin. At time $t = 0\,\text{sec}$, service load 1 and service load 2's cooling water requirements are 1.27kg/sec and 0.78kg/sec respectively. The summation of those two requirements is more than the resource capacity of 0.8kg/sec. Due to the 2 time step delay, no cooling water is supplied until at the 2nd iteration, when service load 1 temperature and service load 2 temperature have increased to 457.31Kelvin and 407.31Kelvin respectively, while the cooling water requirements have increased to 1.32kg/sec and 0.83kg/sec respectively. The service load 1 priority is superior to the service load 2 priority, so the control system tries to satisfy

the service load 1 requirement first by giving an OPEN command to valve 7 in service load 1 and a CLOSE command to valve 11 in service load 2. Service load 1 is cooled down very quickly and its temperature decreases to 397.53Kelvin at the 3rd iteration after 40 seconds cooling by the actual flow rate 0. 64kg/sec, which is different from the capacity 0.8kg/sec of the whole chiller plant. The reason is that the estimation of the capacity 0.8kg/sec is based on the assumption that all of the valves out of the chiller plant are open, but valves in service load 2 are closed in the above situation. However, this difference does not have significant impact on the control system performance in this application. Service load 2's temperature keeps increasing to 409.74Kelvin and its corresponding cooling water requirement increases to 0.85kg/sec while service load 2's cooling water requirement decreases to 0.63keg/sec at the 3rd iteration. Now, service load 1's requirement is less than the capacity of the chiller plant, so the control system tries to redistribute the left cooling water to service load 2 after providing enough cooling water to service load 1. In this application, the control system selects a state of a component proportionally to its state likelihood. Although a state likelihood is very small, it still has the chance to be selected. This reasoning explains the sudden jumps shown in the results of this application. At time $t = 440\,\text{sec}$, valve7 becomes STUCKCLOSE. The distributed Bayesian inference engine detects the valve7 state change at time $t = 560\,\text{sec}$. At time $t = 440\,\text{sec}$, an OPEN command is given to valve7 from the control system; at time $t = 480\,\text{sec}$, the fluid network gives the valve open degree information back to the control system; through data processing, at time $t = 520\,\text{sec}$, the inference engine embedded in this control system gives the information that valve7 is STUCKCLOSE with probability close to 1. At the same time, an OPEN command is sent by the control system to valve8 as a backup valve in the fluid network system and the valve8 state distribution is changed to OPEN with probability close to 1 at time $t = 560\,\text{sec}$. The valve11 state becomes STUCKCLOSE at time $t = 840\,\text{sec}$. However, the inference engine can not detect the damage state of valve 11. In this scenario, valve11 open degree

is not observable. From the state distribution of valve11 shown in Figure 170, we can see that the estimation from the inference engine is not correct; the control system uses this incorrect information and keeps giving valve11 an OPEN command as shown in Figure 171; valve11 can not be open and the service load 1 actual flow rate becomes zero since time $t = 840\,\text{sec}$ as shown in Figure 174; the service load 1 temperature keeps increasing after time $t = 840\,\text{sec}$ as shown in Figure 175. In summary, without enough observations, MSDBNs can not detect some component state changes. Another reason for the wrong estimation in this application for this scenario is that the fluid network is a recycled cooling system and Bayesian network can not handle directed cycles. In the simulation, the directed cycle is broken by giving a hard evidence to a possible measurable flow rate point. It indicates that the recycled cooling system is not the best example to show the effectiveness of MSDBNs inference engine and MSDBNs could perform better for a non-recycled system.

6.2.6 Implementation of the Proposed Methodology to the Notional Ship Chilled Water System

As described in section 6.1.2, the chilled water distribution system on a ship is a good example of large-scale complex systems. It is distributed over the entire ship and it exhibits global behaviors. In addition, its environment is uncertain, especially under combat situations. Currently, this system is primarily operated and managed manually. Since manning is a considerable portion of the ownership cost of US Navy ships, manual control of these highly complex systems directly impacts the cost of supporting these assets in a significant manner. For this reason, automating the control of the chilled water system and empowering the sailors to be able to operate it more efficiently is an important goal if one is to attempt to reduce ownership costs of Navy ships. Herein, based on the general warship configurations, a notional ship chilled water system model is used to test the proposed methodology.

6.2.6.1 Step 1: Check System Constraints & Requirements

The subsystem layout of the notional ship and the corresponding name conventions in the chilled water system are shown in Figure 178. The entire notional ship includes four zones: zone 1, zone 2, zone 3 and zone 4. Zone 1 includes the integrated electric drive subsystem and the chiller-pump network subsystem (two chiller-pump units); zone 2 includes the engine room; zone 3 includes the sonar system and the weapon subsystem; zone 4 includes the radar subsystem and the crew room subsystem. Based on the subsystem distribution, the chilled water system sketch is shown in Figure 179, where each service load corresponds to equipment or an electrical component that needs cooling.

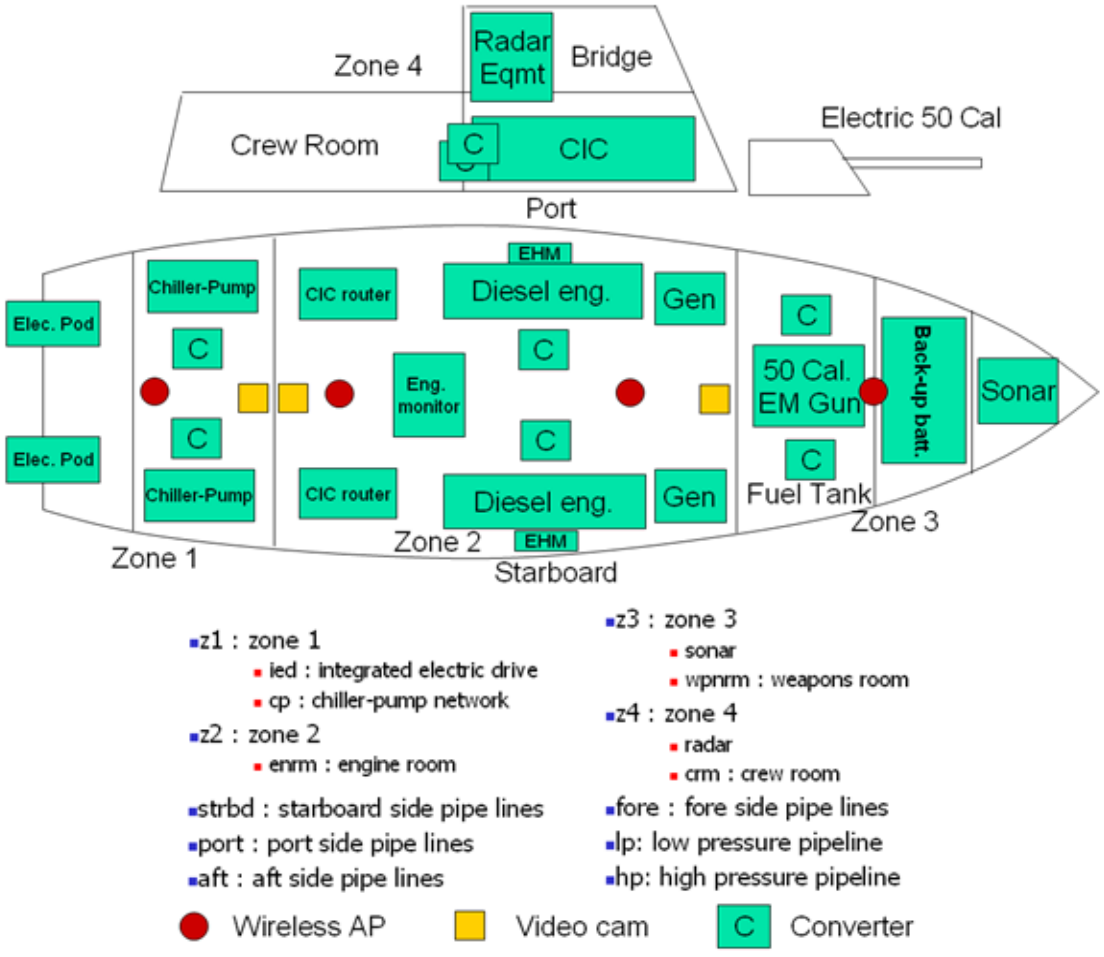


FIGURE 178 THE NOTIONAL SHIP SUBSYSTEM LAYOUT AND NAME CONVENTIONS
(NOTE: THIS FIGURE IS MODIFIED FROM A FIGURE DRAWN BY MY COLLEAGUE KYUNGJIN MOON.)

The system level requirements for the control system for the notional ship chilled water system are as the same as those described in section 6.2.4.1 for the simplified chilled water system and restated as follows:

- Automation: the system can run with less human intervention.
- Robust: the system can run when the information for the control system is contaminated.
- Reconfiguration: the system can reconfigure itself when parts of the system are damaged.
- Dynamics and Flexibilities: the system can adapt to changes in the environment over time.

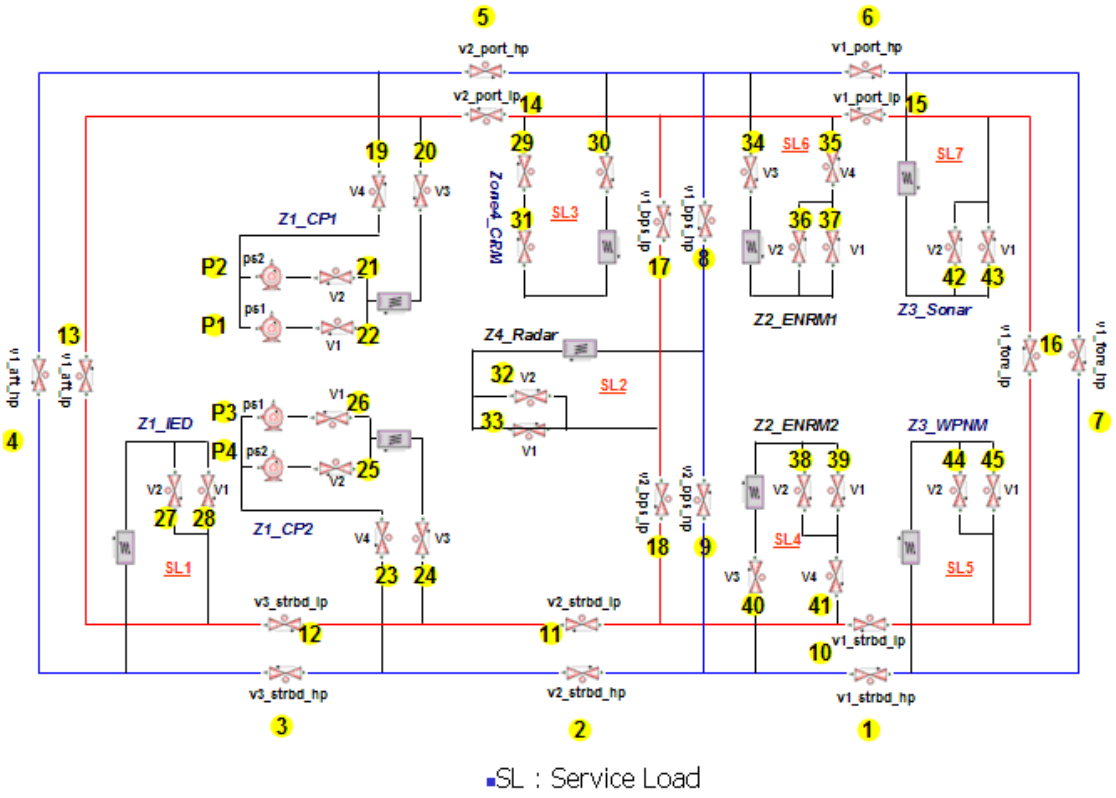


FIGURE 179 THE NOTIONAL SHIP CHILLED WATER SYSTEM SKETCH

6.2.6.2 Step 2: Decompose System

The notional ship chilled water system is decomposed hierarchically into module combinations. The entire system at the high level is decomposed into nine subsystems: chiller-pump network 1, chiller-pump network 2, service load 1, service load 2, service load 3, service load 4, service load 5, service load 6, and service load 7. The subsystem structures are similar to those decribed in section 6.2.4.2 for the simplified chilled water system.

6.2.6.3 Step 3: Establish Control Architecture and Control agents

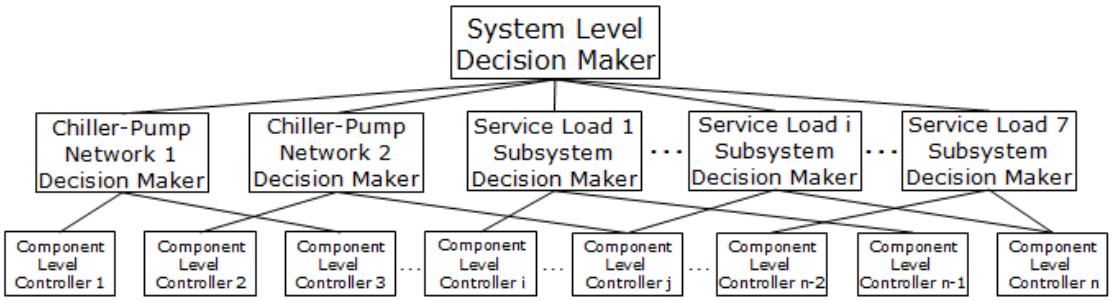


FIGURE 180 CONTROL ARCHITECTURE OF THE NOTIONAL SHIP CHILLED WATER SYSTEM

According to the decomposition, the entire system has three control levels as shown in Figure 180: system level decision maker at the highest level, subsystem decision makers (chiller-pump network 1 decision maker, chiller-pump network 2 decision maker, service load 1 subsystem decision maker, service load 2 subsystem decision maker, service load 3 subsystem decision maker, service load 4 subsystem decision maker, service load 5 subsystem decision maker, service load 6 subsystem decision maker, and service load 7 subsystem decision maker) at the mid level and component level controllers at the lowest level. The system level agent, subsystem agents and component agents have different internal logic, objectives, and information flow, which are similar to those described in

section 6.2.4.3 for the simplified chilled water system. All of the agents established in JADE for the notional ship chilled water control system are shown in Figure 181.

6.2.6.4 Step 4: Establish Sub System Bayesian Networks

The component distribution of the notional ship chilled water system with two chiller-pump units and 7 service loads is shown in Figure 179 and the possible observable flow rate nodes distributed in the fluid network are shown in Figure 182. The entire Bayesian network for the whole system is shown in Figure 183.

From Figure 183, we can see that the structure of the monolithic Bayesian network is too complex and cumbersome for the notional ship chilled water system. Thus, there are four main reasons to establish a distributed probabilistic inference engine for the notional ship chilled water system:

- The fluid network is highly distributed over the entire ship physically.
- Computational intensity for a monolithic Bayesian of a complex system is high.
- A monolithic Bayesian network suffers from a single point failure and the system reliability is reduced.
- Communication cost is high due to the gathering of the distributed environment information to a central processor.

By using MSDBNs, the entire notional ship chilled water system is decomposed into 9 different subsystems. Each subsystem has a corresponding sub Bayesian network. The decomposition and connections among the subsystem Bayesian networks are shown in Figure 184. The structures of the chiller-pump unit 1 sub Bayesian network, the chiller-pump unit 2 sub Bayesian network and the service load 1 sub Bayesian network as three typical sub Bayesian networks are shown in Figure 185 and Figure 186 respectively.

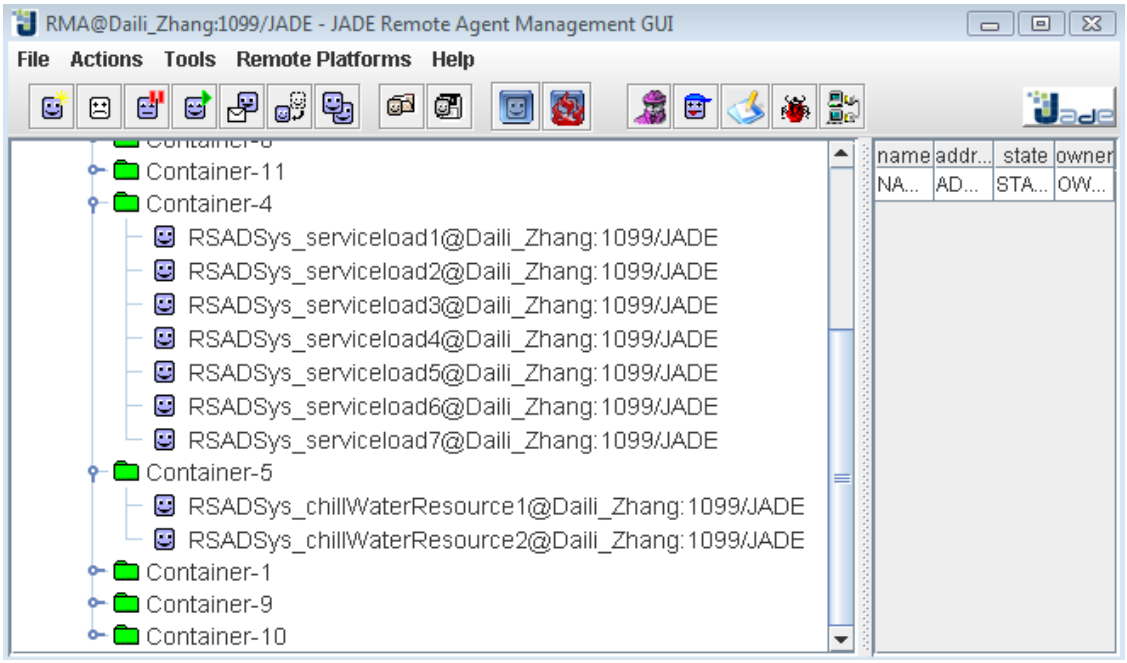


FIGURE 181 AGENTS OF THE NOTIONAL SHIP CHILLED WATER CONTROL SYSTEM ESTABLISHED IN JADE

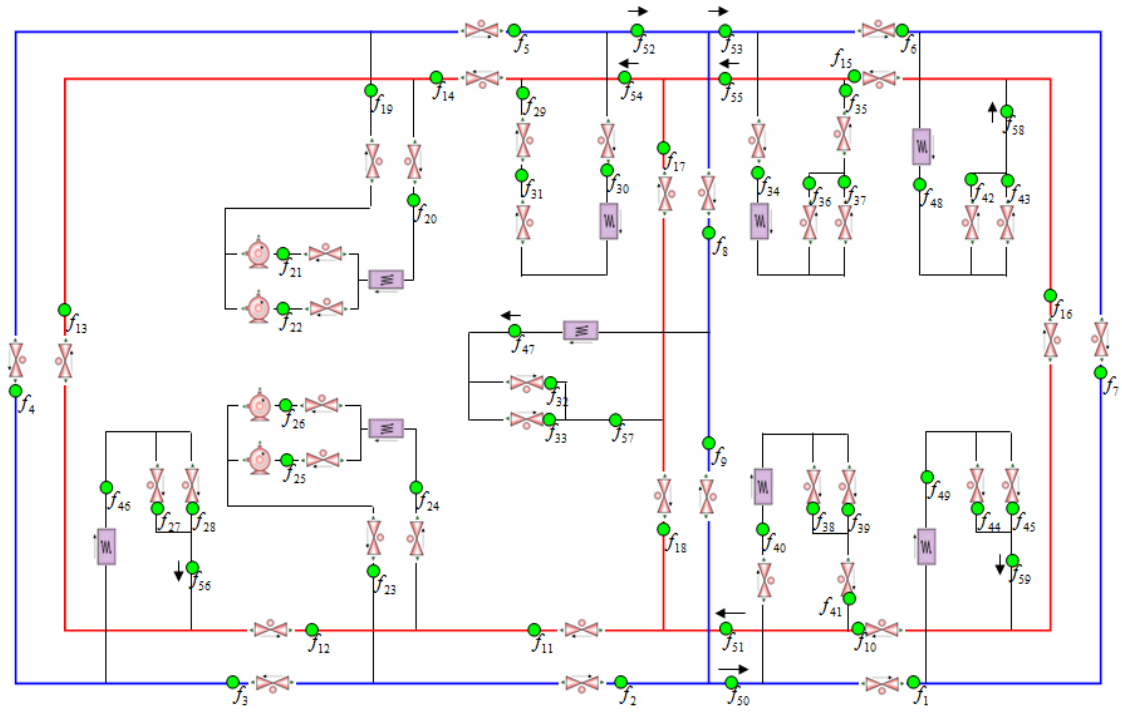


FIGURE 182 POSSIBLE OBSERVABLE FLOW RATE NODE DISTRIBUTION IN THE NOTIONAL SHIP CHILLED WATER SYSTEM

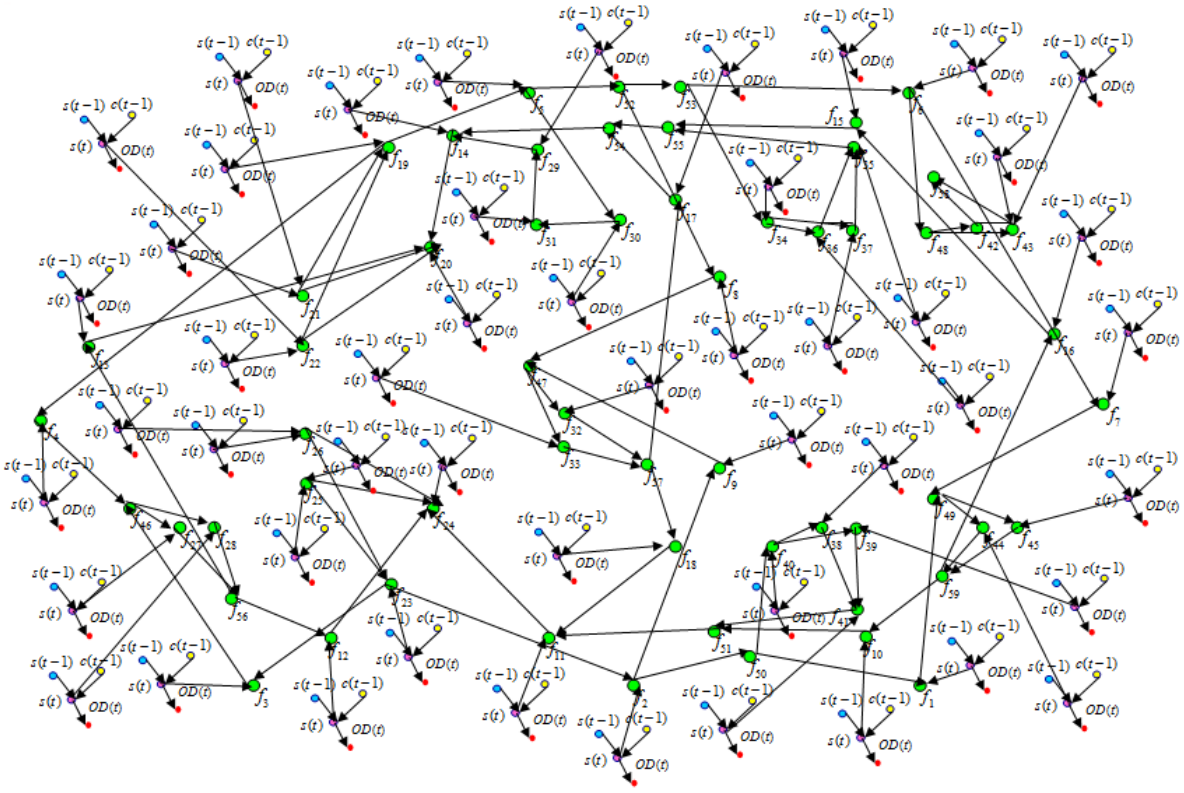


FIGURE 183 BAYESIAN NETWORK FOR THE WHOLE NOTIONAL SHIP CHILLED WATER SYSTEM

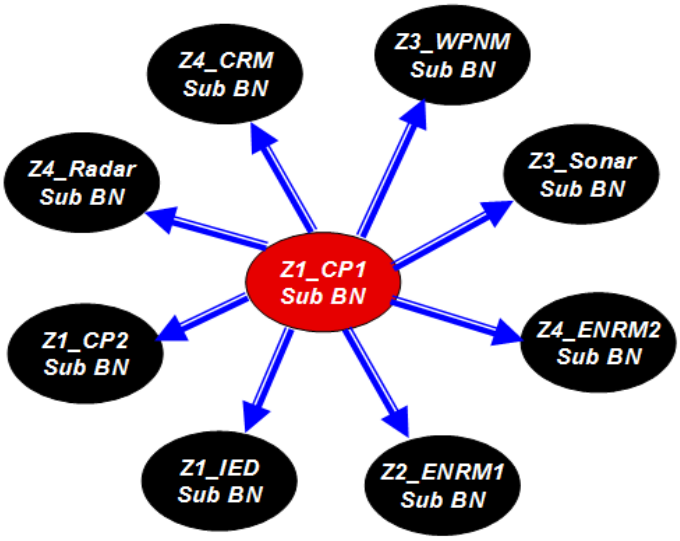


FIGURE 184 BAYESIAN NETWORK DECOMPOSITION AND CONNECTION FOR THE WHOLE NOTIONAL SHIP CHILLED WATER SYSTEM

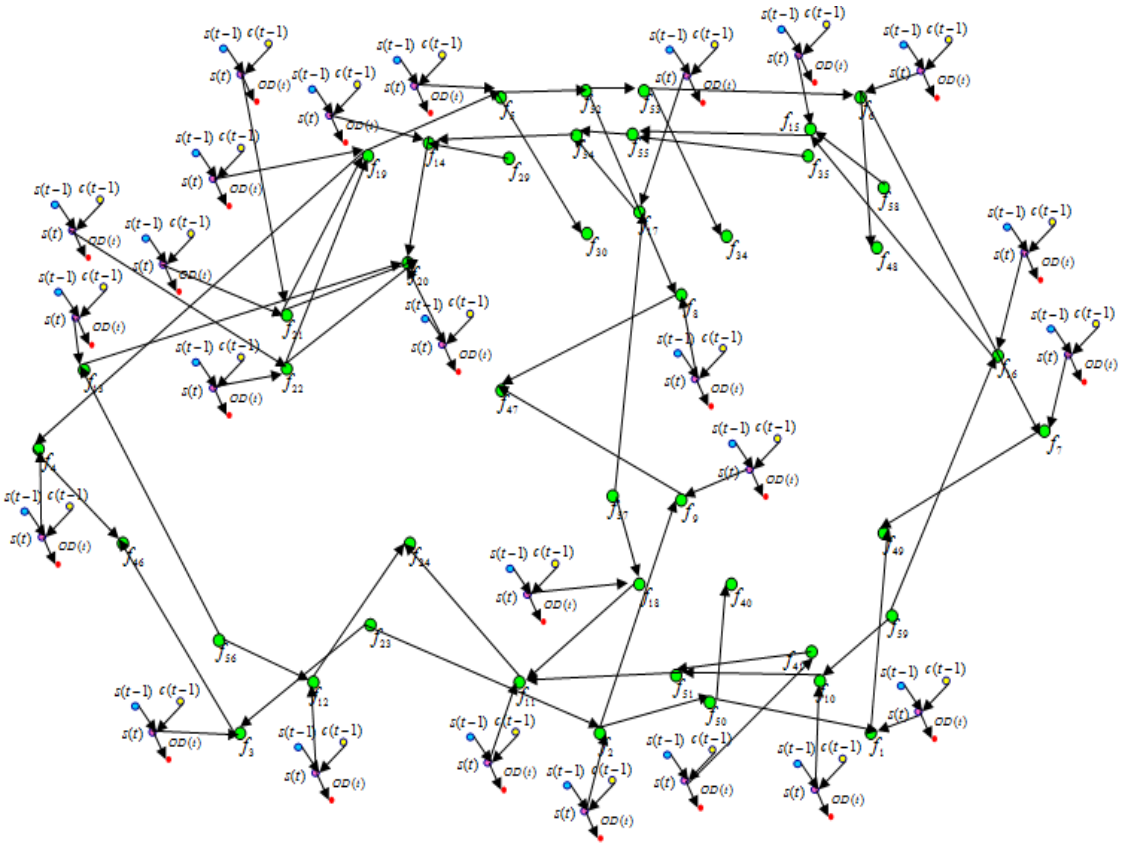


FIGURE 185 SUB BAYESIAN NETWORK OF CHILLER-PUMP UNIT 1 OF THE NOTIONAL SHIP CHILLED WATER SYSTEM

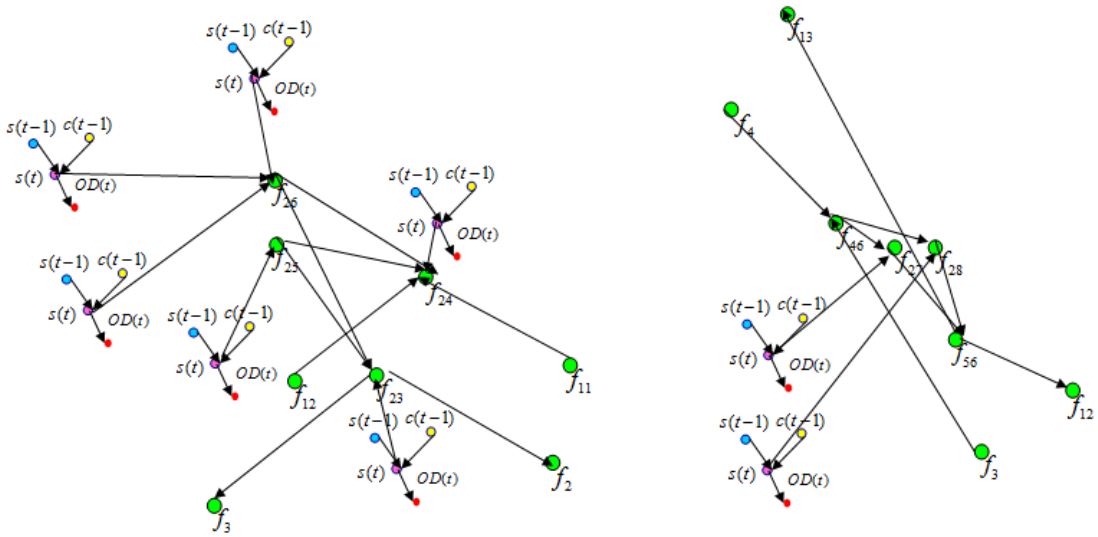


FIGURE 186 SUB BAYESIAN NETWORKS OF CHILLER-PUMP UNIT 2 AND SERVICE LOAD 1 OF THE NOTIONAL SHIP CHILLED WATER SYSTEM

As decribed in section 6.2.6.4 for the simplified chilled water system, a two time-slice homogeneous Dynamic Bayesian Network (DBN) is used for each component state evolution from time t to time t+1. The fully factorized Boyen-Koller (BK) approximation algorithm is used for each local sub Bayesian network belief updating over the time span and the static Junction Forest Linkage Tree (JFLT) algorithm is used for the global system belief updating over the entire network, as decribed in section 6.2.4.4 for the simplified chilled water system.

6.2.6.5 Step 5: Design Each Sub Bayesian Network as an Agent and Design Its Corresponding Internal Logic and Series of Behaviors for DSTO, DDSSS, DRIS and DBP

This step for the notional ship chilled water system is similar to that described in section 6.2.4.5 for the simplified chilled water system. The entire MSDBNs with 9 different sub Bayesian networks need to satisfy the following three conditions to achieve globally consistent inferences as well:

- tree organization,
- d-sep set between two neighbored sub Bayesian networks,
- and the running intersection property.

6.2.6.6 Step 6: Insert MSDBNs into Control Structure

In the previous steps, the control agents and MSDBNs have been established. The current step will insert the MSDBNs into the control structure to make them work interactively. The way that the MSDNBs are inserted into the control structure is as the same as described in section 6.2.6.6 for the simplified chilled water system.

6.2.6.7 Step 7: Verify and Validate the Designed Control System

Now, the whole control system has been established. It is time to verify, validate and iterate the designed control system. The methods used to integrate the Flowmaster model,

the thermo-electric model, the agent based control model with MSDBNs and the scenario definition model into Phoniex Integration's ModelCenter, are as the same as those described in section 6.2.4.7 for the simplified chilled water system.

6.2.7 Results and Discussions

The integrated model for the application described in detail previously is ready to run. As in the simplified chilled water system, a script scheduler written in VBScript controls the running mode, such as when and how to run each Contribution Analysis (CA), when and how to exchange information between different CAs, how many time steps a CA should run, what outputs should be collected and stored, etc.

TABLE 11 LIST OF 3 DIFFERENT SCENARIOS FOR THE NOTIONAL SHIP CWS

Scenario / Item	Scenario 1	Scenario 2	Scenario 3
Component State	all undamaged	all undamaged	valve27 STUCKCLOSE at t=44sec, valve28 STUCKCLOSE at t=84sec, valve32 STUCKCLOSE at t=44sec, valve33 STUCKCLOSE at t=84sec
Flow Rate Observability	all observable	all not observable	only f32, f33 observable
Valve Open Degree Observability	all observable	all observable	valve32 and valve33 not observable, others observable
Resource Capability (kg/sec)	[0.8, 0.8]	[0.8, 0.8]	[0.8, 0.8]
Service Load Initial Temperature (K)	[380 370 360 350 283 293 303]	[380 370 360 350 283 293 303]	[380 370 360 350 283 293 303]
Service Load Incoming Power (kw)	[50 50 50 50 100 100 100]	[50 50 50 50 100 100 100]	[50 50 50 50 100 100 100]
Simulation Time Step (sec)	4	4	4
Simulation Time (sec)	[0, 120]	[0, 120]	[0, 120]

The parameters of the service loads for the power components for differenent scenarios were the same as those listed in Table 8, except for the initial temperature of each service load, which were reset for each scenario. All of the scenarios were defined in different Excel worksheets. All of the outputs from every CA for each scenario were collected and

stored into a corresponding Excel worksheet. Three different scenarios are listed in Table 11 and discussed in detail in the following sections.

6.2.7.1 Scenario 1 of the Notional Ship CWS (Nominal Condition 1):

The conditions of scenario 1 are listed in column 2 of Table 11 and a few monitored outputs are shown from Figure 187 to Figure 194.

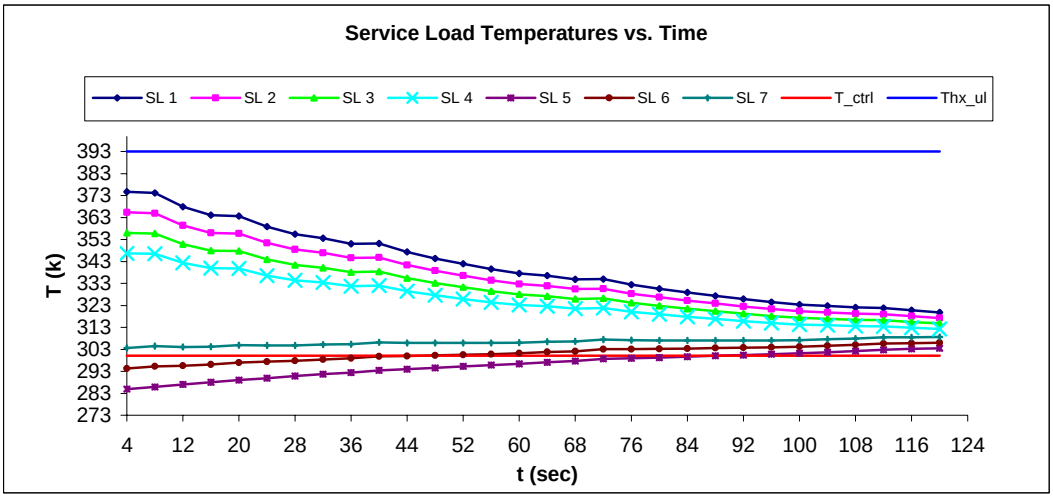


FIGURE 187 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD TEMPERATURES VS.TIME

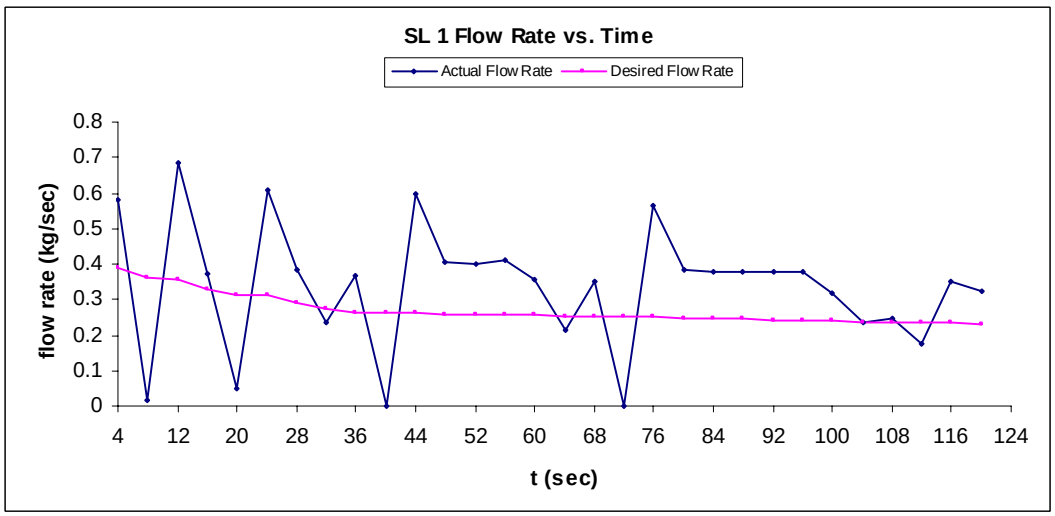


FIGURE 188 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 1 FLOW RATE VS.TIME

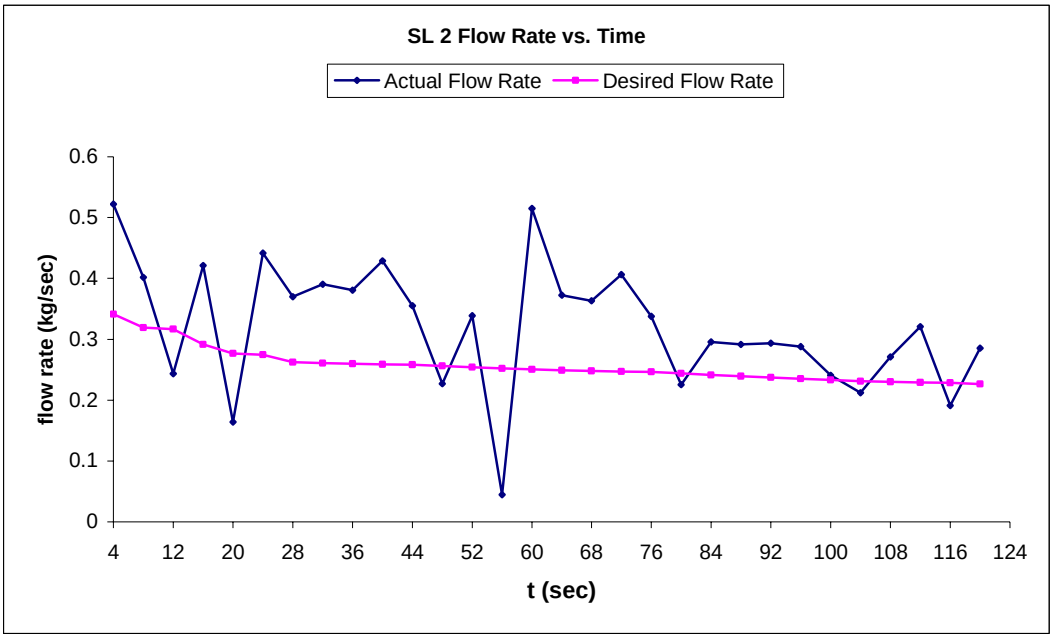


FIGURE 189 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

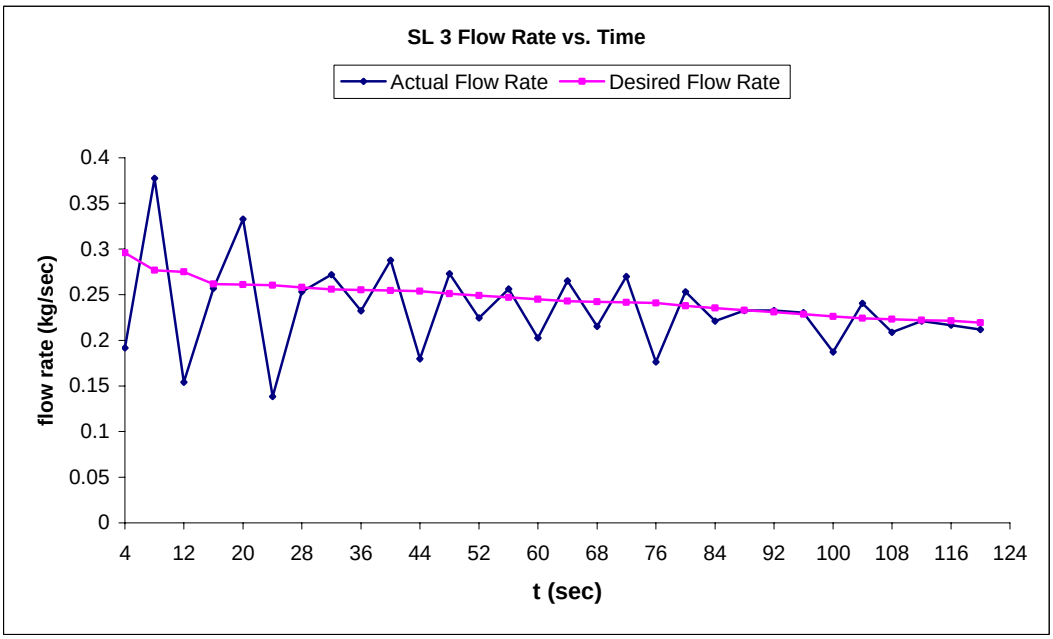


FIGURE 190 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 3 FLOW RATE VS. TIME

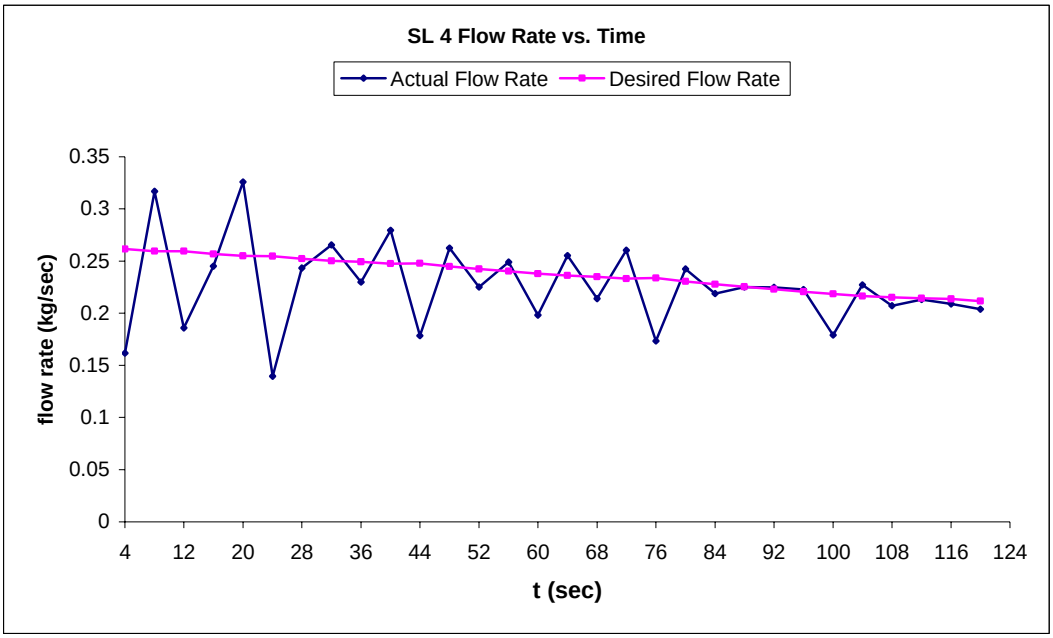


FIGURE 191 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 4 FLOW RATE VS. TIME

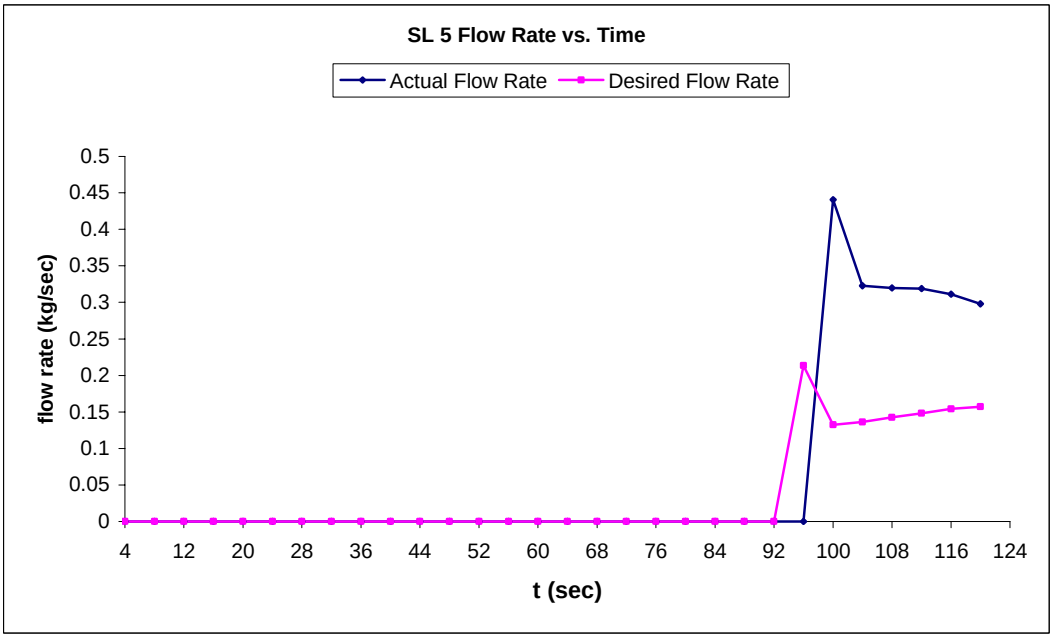


FIGURE 192 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 5 FLOW RATE VS. TIME

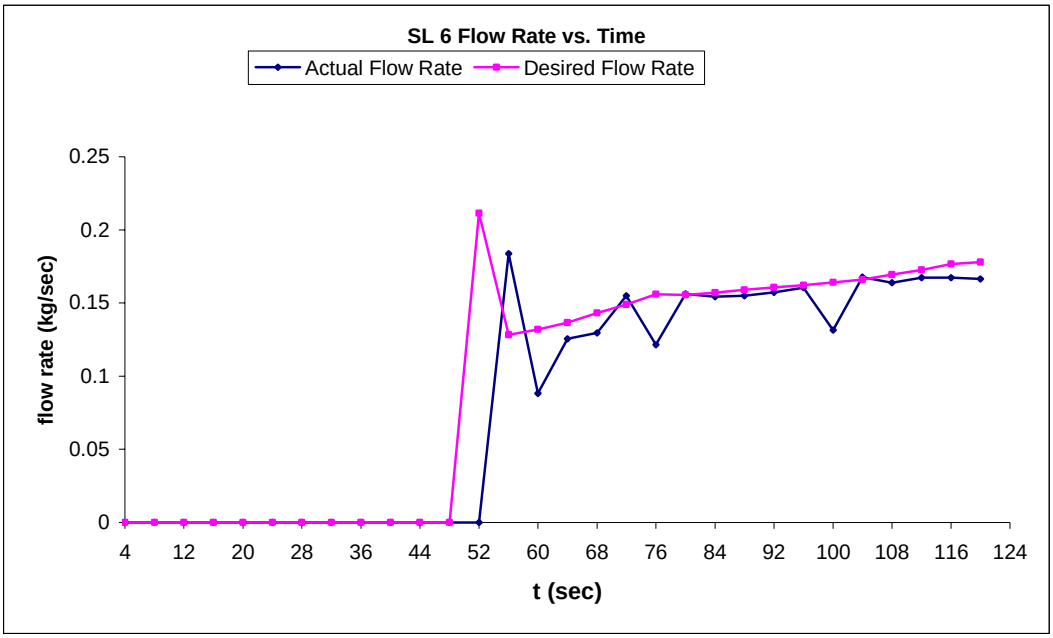


FIGURE 193 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 6 FLOW RATE VS.TIME

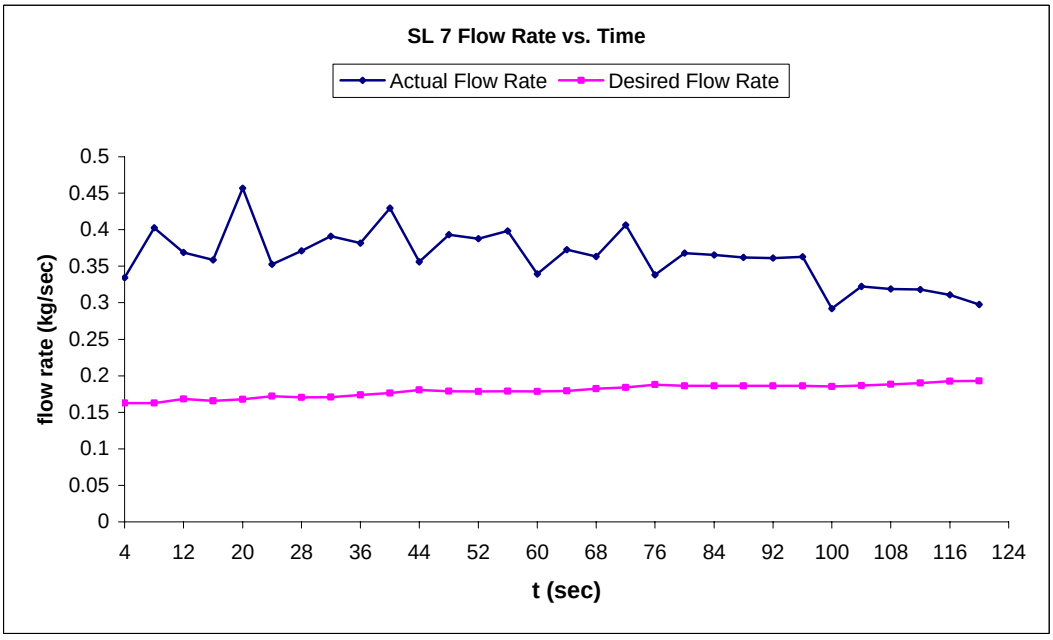


FIGURE 194 SCENARIO 1 (THE NOTIONAL SHIP CWS): SERVICE LOAD 7 FLOW RATE VS. TIME

From the initial conditions, at time $t = 0$, the service load 1, service load 2, service load 3, service load 4 and service load 7's temperatures exceed the control temperature, while the service load 5 and service load 6 temperatures are below the control temperature. Therefore, only the service load 1, service load 2, service load 3, service load 4 and service load 7 require cooling water. Their requirement summation is greater than one pump-chiller unit capacity, so both of the two pump-chiller units are working simultaneously. The service load 5 and service load 6 as power components, each has 100kw incoming power and the efficiency of the incoming power is 0.7, so 30 percent of the incoming power dissipates into the service load 5 and the service load 6, and causes their temperatures to increase. Thermo-Electrical System (TES) CA calculates the required cooling fluid flow rates according to the service load current temperatures every 4 seconds. At time $t = 96\,\text{sec}$, the temperature of service load 5 exceeds the control temperature as shown in Figure 187 and its desired flow rate is greater than zero as shown in Figure 192. Similarly, at time $t = 48\,\text{sec}$, the service load 6 temperature exceeds the control temperature as shown in Figure 187 and its desired flow rate is greater than zero as shown in Figure 193. From Figure 188 to Figure 194, we can see that the actual flow rate for each service load does follow the trend of its required flow rate, which indicates that the control system gets the correct inferences from the MSDBNs inference engine and gives the correct control strategies for controlling the entire chilled water system. In summary, for the nominal case, the MSDBNs inference engine provides correct inferences, and the control system makes the right decisions and distributes the resource to different service loads accordingly.

6.2.7.2 Scenario 2 of the Notional Ship CWS (Nominal Condition 2):

The conditions of scenario 2 are listed in column 3 of Table 11, and the monitored outputs are shown from Figure 195 to Figure 202.

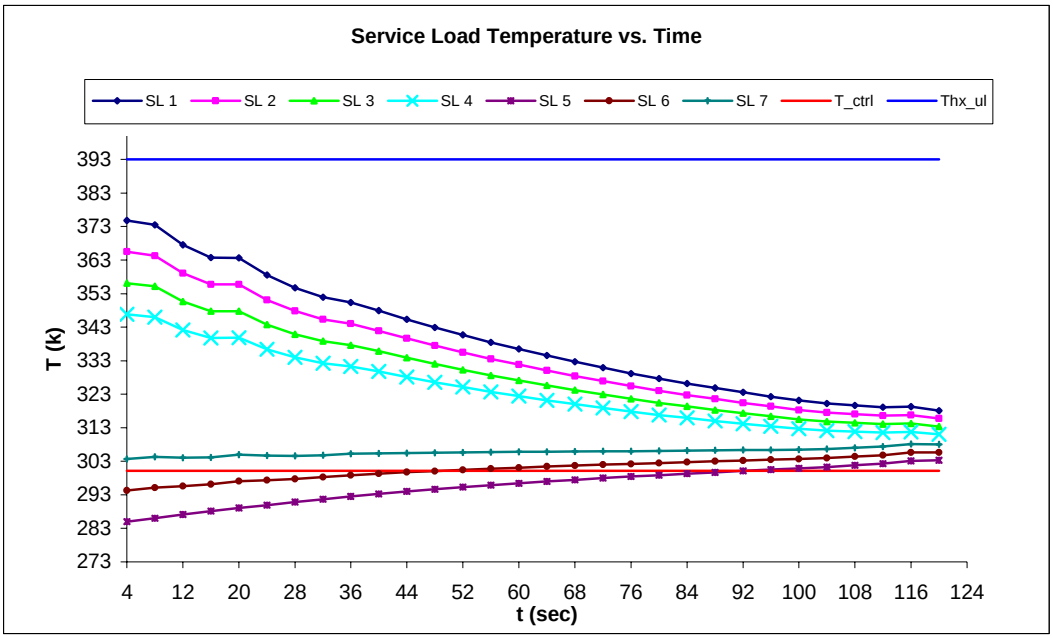


FIGURE 195 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD TEMPERATURES VS. TIME

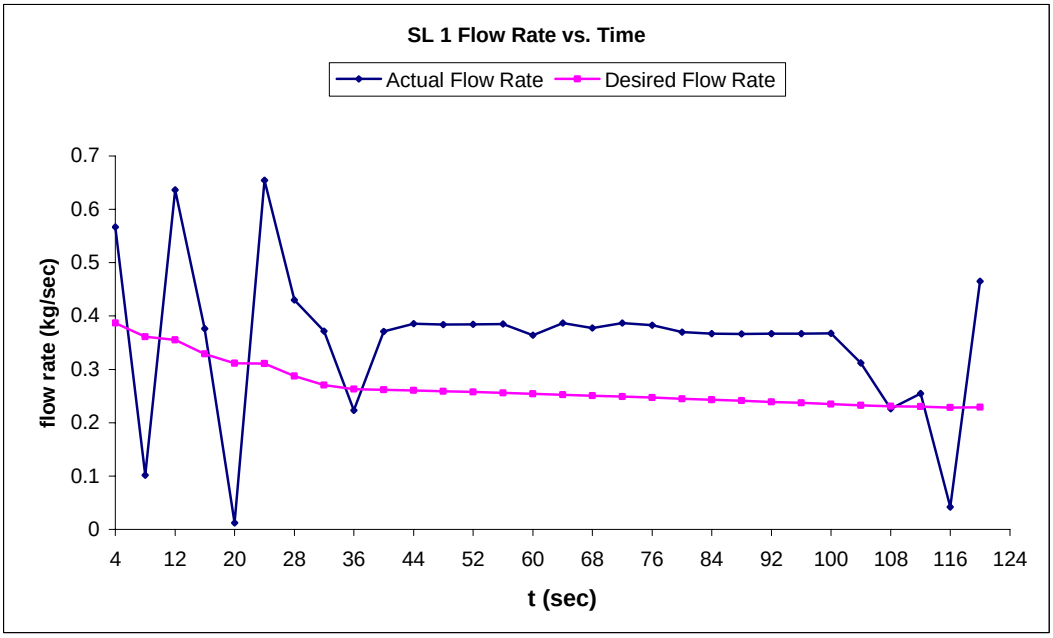


FIGURE 196 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

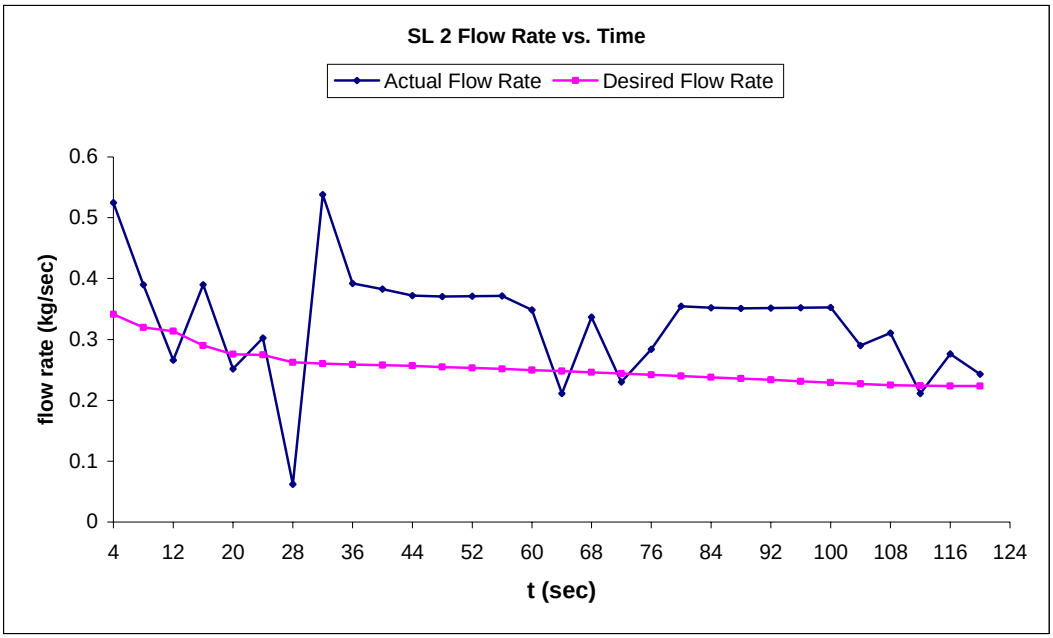


FIGURE 197 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

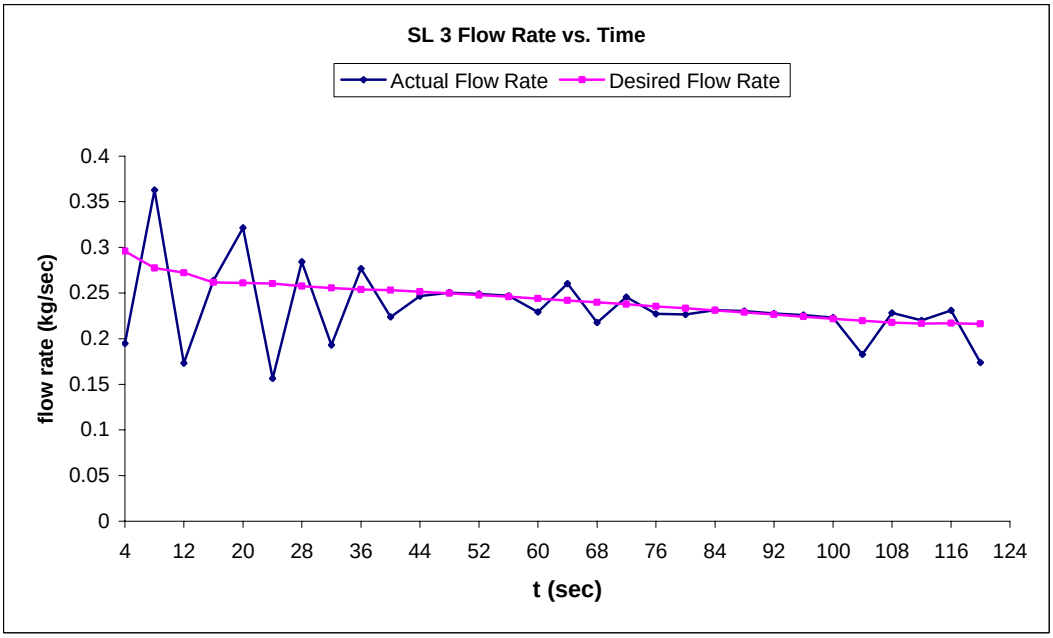


FIGURE 198 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 3 FLOW RATE VS. TIME

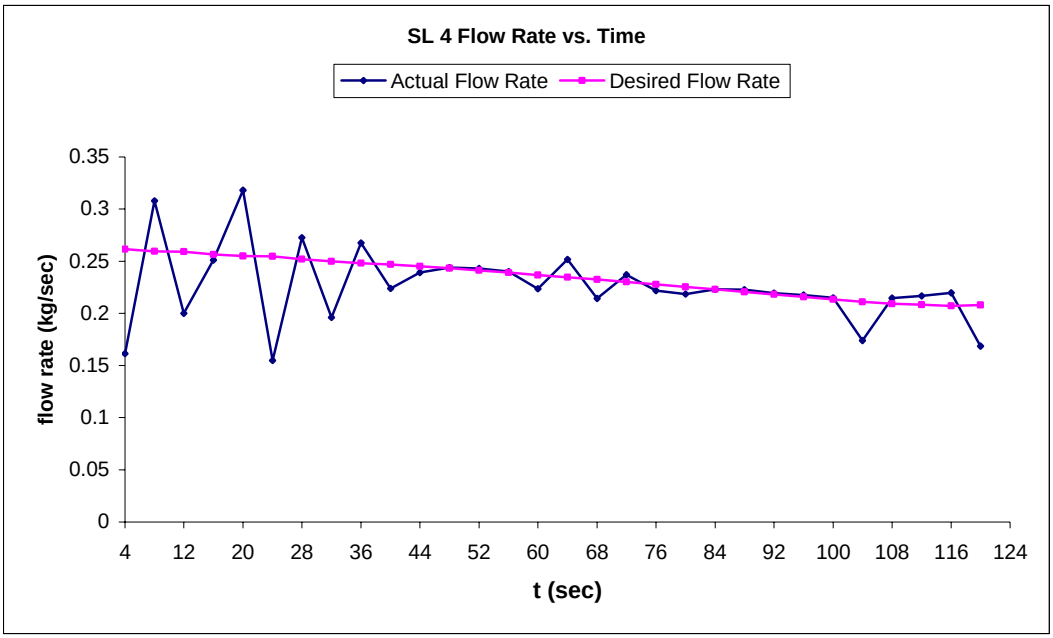


FIGURE 199 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 4 FLOW RATE VS. TIME

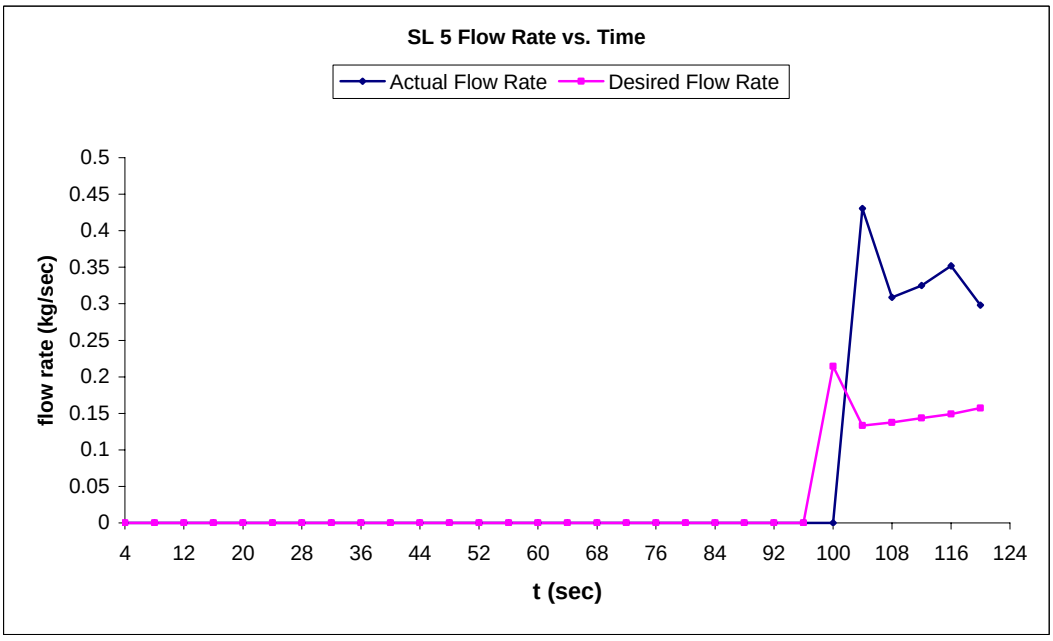


FIGURE 200 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 5 FLOW RATE VS. TIME

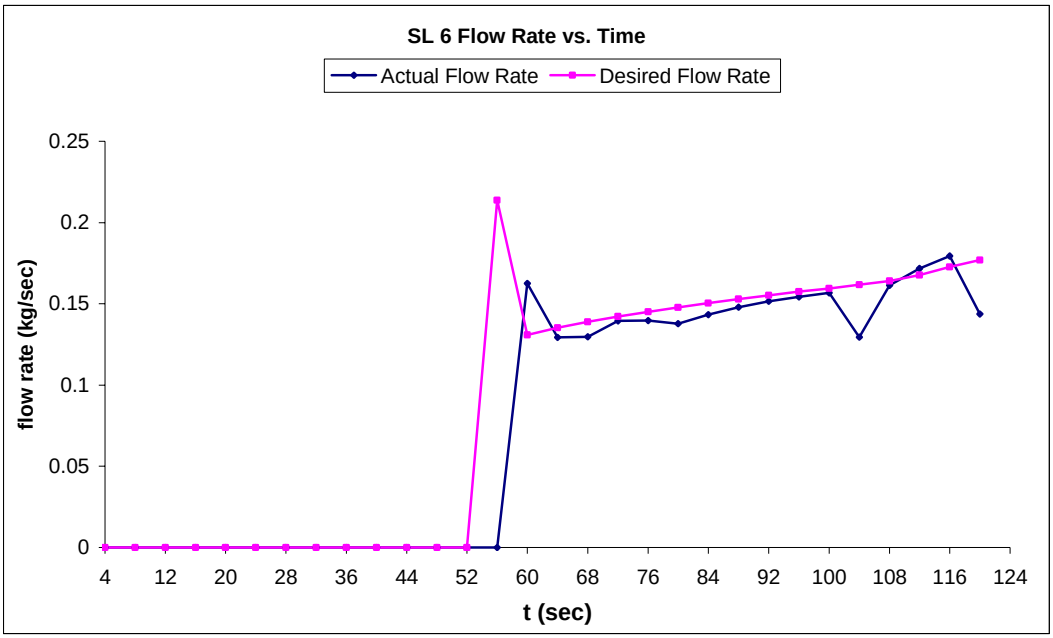


FIGURE 201 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 6 FLOW RATE VS. TIME

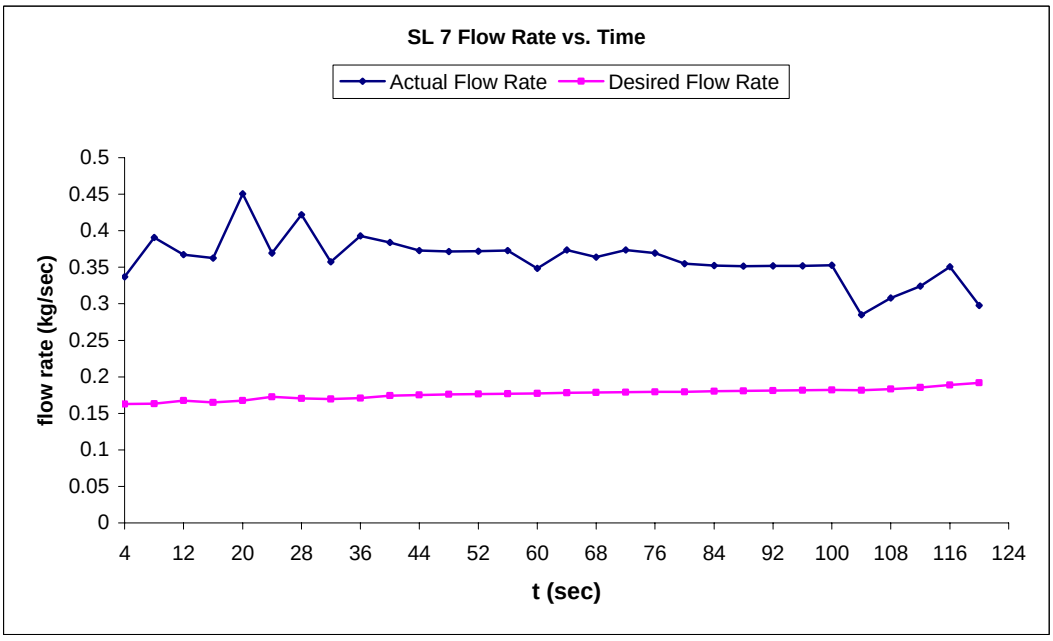


FIGURE 202 SCENARIO 2 (THE NOTIONAL SHIP CWS): SERVICE LOAD 7 FLOW RATE VS. TIME

Scenario 2 and scenario 1 have the same intial conditions and component state changes except that there is no flow rate observation for scenario 2. Comparing the results shown from Figure 195 to Figure 202 for scenario 2 with the results shown from Figure 187 to Figure 194 for scenario 1, we can see that the results in scenario 2 are similar to the results in scenario 1. In summary, without the flow rate observations, but with the complete valve open degree observations, the MSDBNs inference engine provides correct inferences for the component states and the control system makes the correct control strategies for controlling the entire notional ship chilled water system.

6.2.7.3 Scenario 3 of the Notional Ship CWS (Nominal Condition 2):

The conditions of scenario 3 are listed in column 4 of Table 11 and the monitored outputs are shown from Figure 203 to Figure 210.

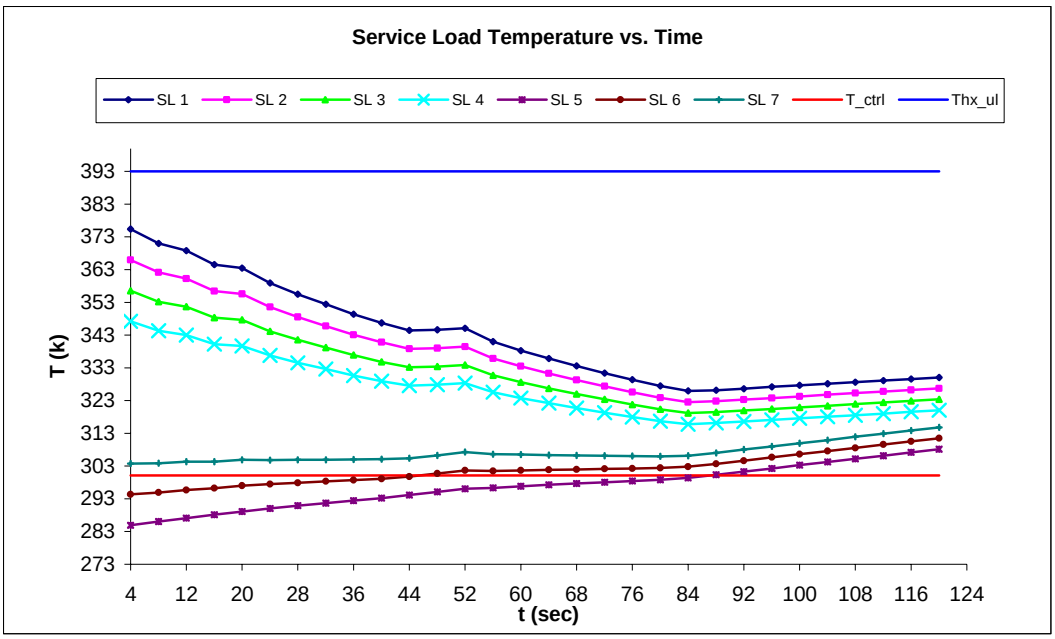


FIGURE 203 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD TEMPERATURES VS. TIME

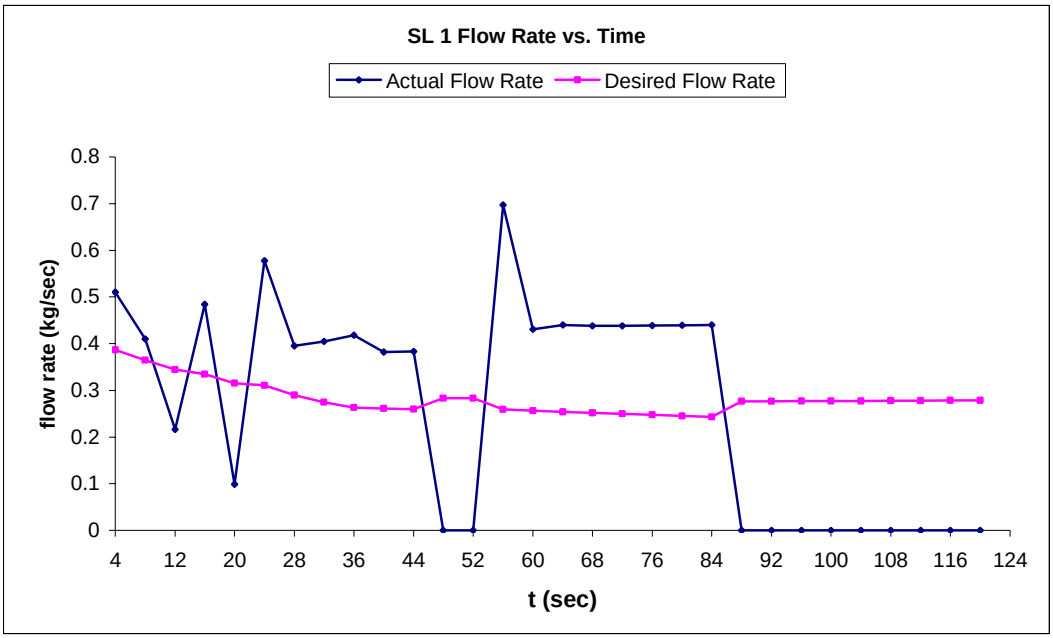


FIGURE 204 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 1 FLOW RATE VS. TIME

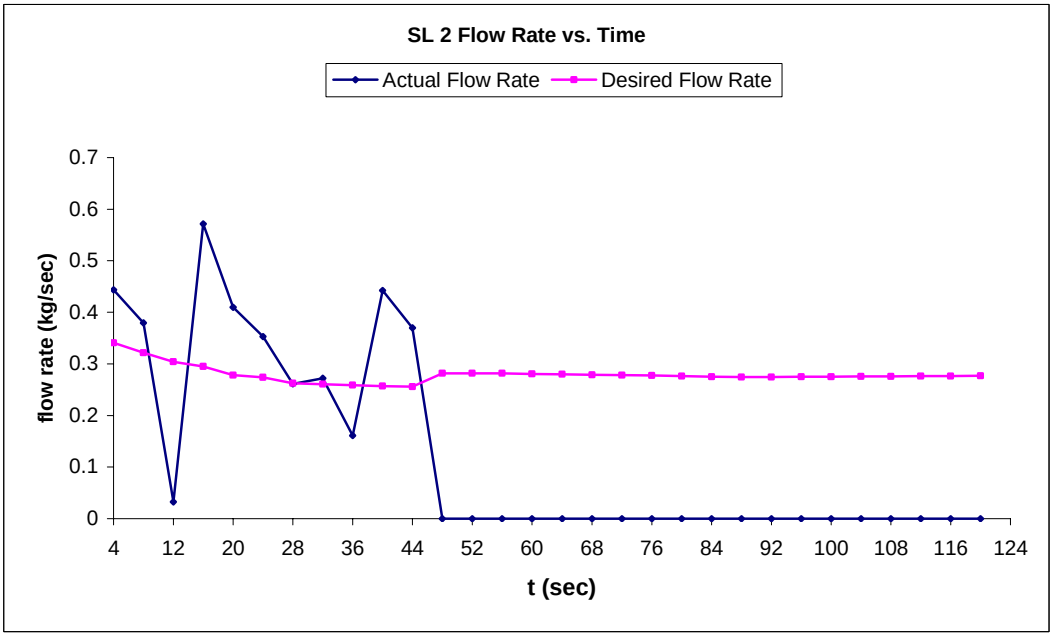


FIGURE 205 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 2 FLOW RATE VS. TIME

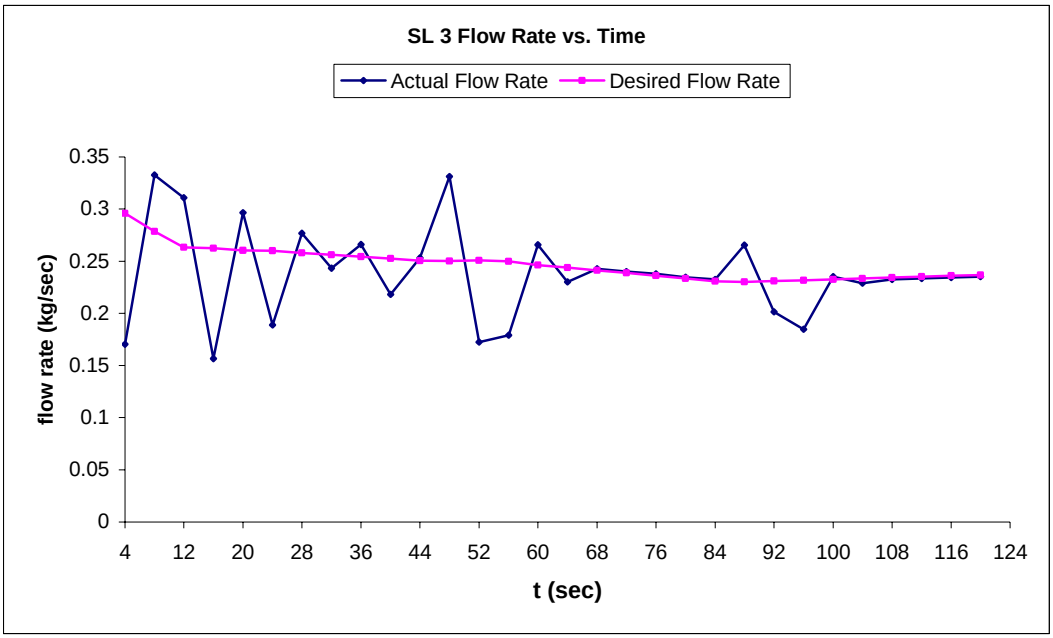


FIGURE 206 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 3 FLOW RATE VS. TIME

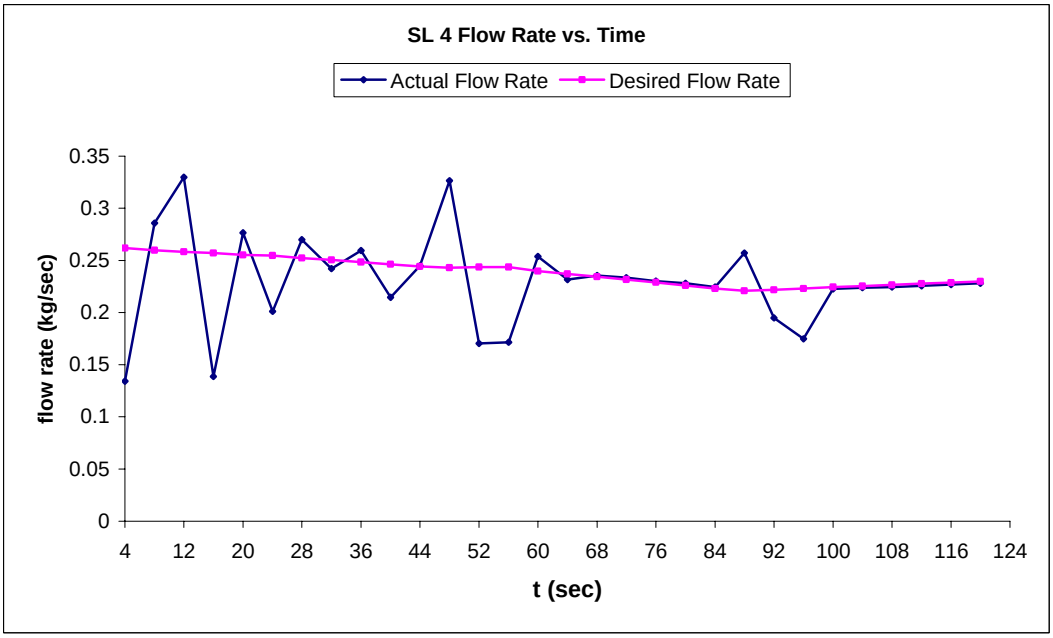


FIGURE 207 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 4 FLOW RATE VS. TIME

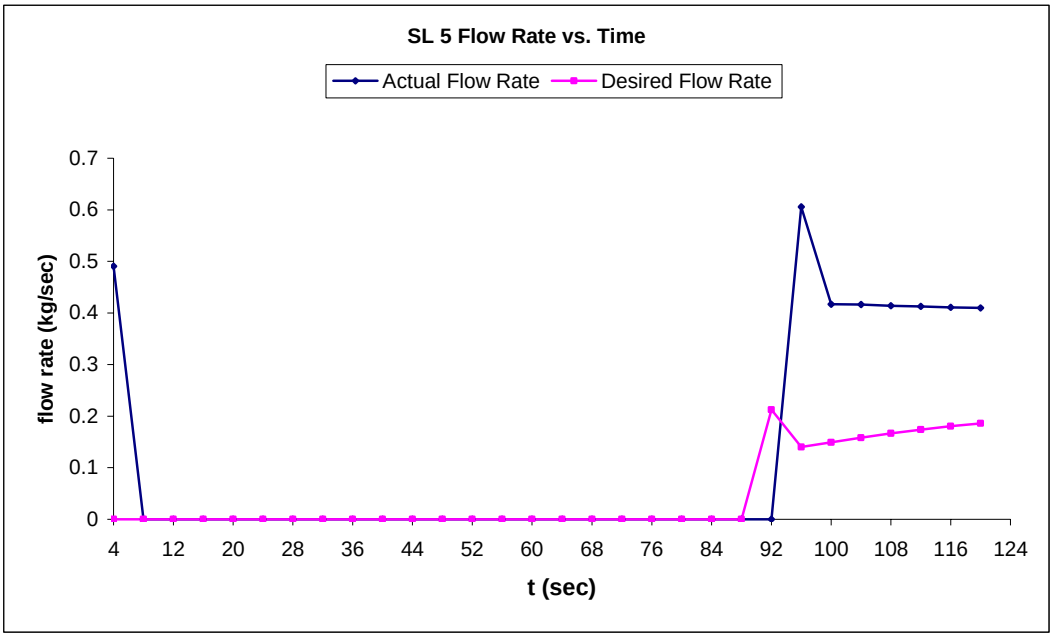


FIGURE 208 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 5 FLOW RATE VS. TIME

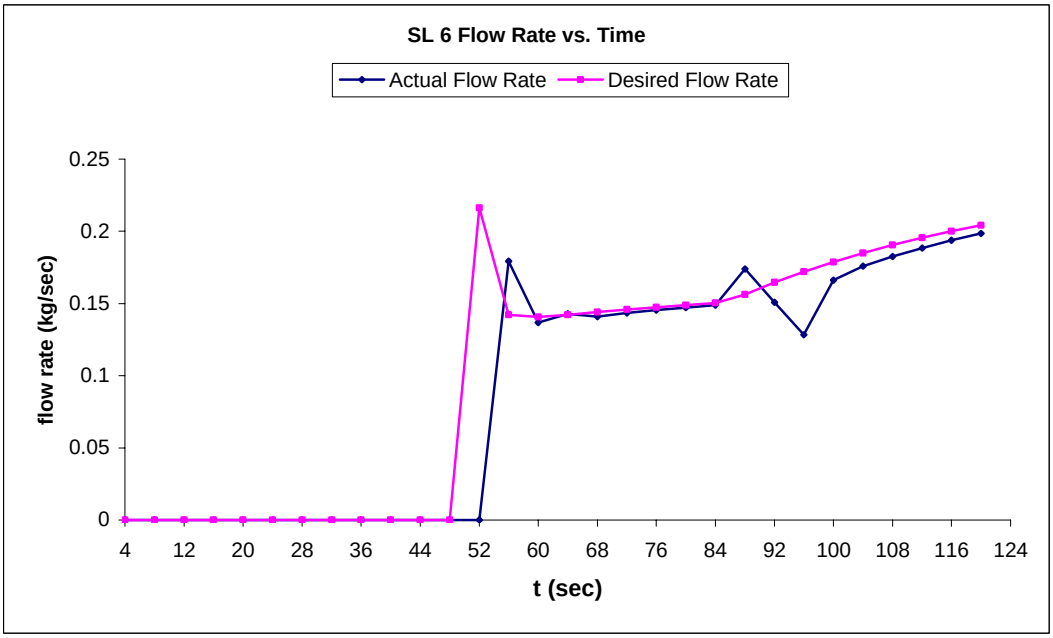


FIGURE 209 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 6 FLOW RATE VS. TIME

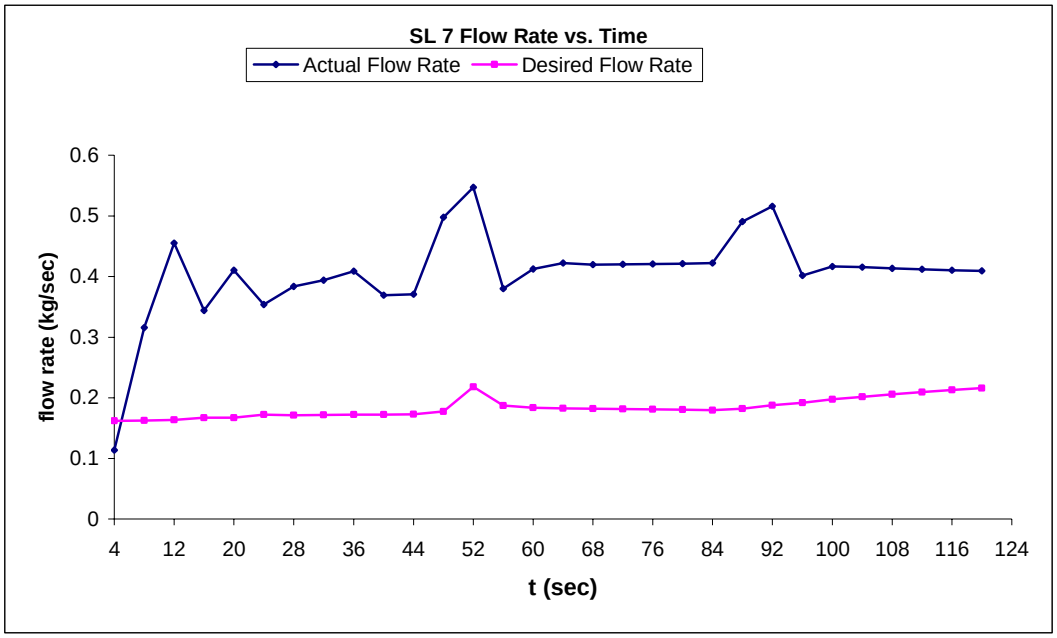


FIGURE 210 SCENARIO 3 (THE NOTIONAL SHIP CWS): SERVICE LOAD 7 FLOW RATE VS. TIME

As indicated in column 4 of Table 11, valve27 and valve32 become STUCKCLOSED at time $t = 44\,\text{sec}$, and valve28 and valve33 become STUCKCLOSED at time $t = 84\,\text{sec}$. Valve27 and valve28 have valve open degree observations but no flow rate observations, while valve32 and valve33 have flow rate observations, but no valve open degree observations. From Figure 204, we can see that, for the service load 1, the MSDBNs can detect that valve27 becomes STUCKCLOSED, and provide the correct inferences to the control system and the control system gives the correct control commands to switch to valve28 and the service load 1 keeps getting chilled water until valve28 becomes STUCKCLOSED. After valve28 becomes STUCKCLOSED at time $t = 84\,\text{sec}$, the service load 1 cannot get chilled water as shown in Figure 204 and its temperature is increasing, as shown in Figure 203. By contrast, from Figure 205, the MSDBNs cannot dectect that valve32 becomes STUCKCLOSED and it provides wrong inference information to the control system and the control system never switches to valve33 for the service load 2 to get chilled water, and its actual flow rate becomes zero from time

$t = 44\,\text{sec}$, as shown in Figure 205 and its temperature keeps increasing after time $t = 44\,\text{sec}$, as shown in Figure 203. The other five service load desired flow rates follow their corresponding actual flow rates, as shown from Figure 206 to Figure 210. In summary, without enough and appropriate observations, MSDBNs can not detect some component state changes. Similar to the scenario 5 for the simplified chilled wate system, the reason for the wrong estimation in the notional ship chilled water system application for this scenario is that the fluid network is a recycled cooling system and Bayesian network can not handle directed cycles. In the simulation, the directed cycle is broken by giving a hard evidence to a possible measurable flow rate point. It indicates that the recycled cooling system is not the best example to show the effectiveness of MSDBNs inference engine and MSDBNs could perform better for non-recycled systems.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

7.1 Revisit Research Objectives and Research Questions

The general research objective of this dissertation is to develop a comprehensive, generalized framework for the control system design of large-scale complex systems under significant uncertainties. This general objective is decomposed into two interactive branches: distributed inference engine design and distributed control architecture design. Based on that decomposition, three sub research objectives follow naturally:

- ***O1: Establish a general control architecture for large-scale complex systems, that can provide the capability of being robust, flexible, reusable, and scalable.***
- ***O2: Establish an inference engine that can handle incomplete and noisy information, and make inference reasoning automatically, efficiently and robustly. This inference engine should have the capability of reaching global state consistency under some conditions and at the same time, keeping privacy of subsystem domains, i.e., each subsystem revealing partial information to the public or its concerned neighbors.***
- ***O3: Integrate the inference engine into the control architecture to make them work smoothly.***

Corresponding to the three research objectives, two general research questions arise:

- ***Q1: Is there a distributed control architecture that can provide the capabilities of being robust, scalable, flexible and reusable for controlling large-scale complex systems with limited communications?***

- ***Q2: Is there an inference engine that can handle uncertainties of large-scale complex systems?***

Question 2 can be decomposed into three sub questions:

- ***Q2.1: How does the inference engine handle incomplete and uncertain data sets?***

- ***Q2.2: How does the inference engine work for a distributed dynamic system?***

- ***Q2.3: How does the inference engine reach global consistencies if it is distributed?***

7.2 Hypotheses and Contributions

In order to provide anwsers to these research questions, first, a literature review was conducted to investigate current existing methods. For controlling a large-scale complex system, numerous agents need to work cooperatively to achieve one or more global control goals; this is called Multi-Agent Based Control (MABC). Basically, MABC is implemented by decomposing a complex system into many smaller parts and each part is treated as a relatively isolated agent with the responsibility of controlling a local region intelligently and interacting with other parts in the whole system. MABC was chosen as the base for the methodology and process in this research. MABC, as an implementation of the modern distributed control framework discussed in Chapter II, has several prominent advantages for controlling large-scale complex systems. A new control architecture, Hybrid Multi-Agent Based Control (HyMABC) architecture was proposed to address the challenges of controlling large-scale complex systems.

As proposed solutions to research question 1, hypothesis 1.1 and hypothesis 1.2 were presented as follows:

- ***H1.1: Hybrid distributed multi-agent based control architecture is scalable, flexible, and reusable.***

- ***H1.2: By using replication logical rings for critical agents, a hybrid distributed multi-agent based control architecture provides robustness of a control system to partial damage.***

HyMABC architecture combines hierarchical control architecture and module control architecture with logical replication rings. First, it decomposes a complex system hierarchically; second, it combines the components in the same level as a module and designs common interfaces for all of the components in the same module; third, a few replications are made for critical agents and are organized into logical rings. It keeps clear guidelines for complexity decomposition and also reduces communication complexity.

Replication rings are used to improve robustness of the system. The basic idea of replication rings is that a few replications are made for critical agents and are organized into a ring structure. Agents attached to a critical component/subsystem can be distributed to different locations. One replication can detect the states of its neighbored replications. If it detects that its neighbor is not available, it will broadcast this information to other replications, and the remaining replications will be reorganized into a new ring. The agents attached to the unavailable replication become orphans and reattach themselves to any other replications.

As proposed solutions to research question 2, hypothesis 2.1, hypothesis 2.2 and hypothesis 2.3 are presented as follows:

- ***H2.1: If Dynamic Bayesian Networks (DBNs) can be established by investigating cause-effect relationships among different variables, incomplete and uncertain information can be handled systematically and efficiently.***

- ***H2.2: If Multiple Sectioned Dynamic Bayesian Networks (MSDBNs) can be established for a large-scale complex system,***

  - ***each sub Bayesian network can make its own decisions for its local system relatively independently;***

  - ***globally consistent inferences can be made by each sub Bayesian network through limited message passing among sub Bayesian network agents;***

  - ***sub-Bayesian network agents can self-organize into MSDBNs structures and make inferences automatically in a distributed way when a system is partially damaged.***

- ***H2.3: Bayesian network agents can be embedded into the hybrid control architecture as an internal distributed inference engine to handle system uncertainties.***

The basic idea behind MSDBNs is that they decompose a big knowledge-based system into several agents. Each agent holds its partial perspective of a large problem domain by representing its knowledge as a dynamic Bayesian network. Each agent accesses local evidence from its corresponding local sensors, and communicates with other agents through finite message passing. It reasons about local component states with the local evidences and limited global evidences, and sends this information to its local controller/decision maker. By organizing these distributed agents into certain structures, globally consistent inferences are achievable in a distributed way. By using different frequencies for local DBN agent belief updating and global system belief updating, it balances the communication cost and inference global consistency. In this dissertation,

the fully factorized Boyen-Koller (BK) approximation algorithm was used for local DBN agent belief updating, and the static Junction Forest Linkage Tree (JFLT) algorithm is used for global system belief updating. MSDBNs have several notable advantages for state estimations of large-scale complex systems as follows:

- They provide a coherent framework for probabilistic inference in a large domain.
- They can be applied under a single agent paradigm or a cooperative multi-agent paradigm.
- They support object-oriented inferences.
- They can infer their local node states based on its local available information correctly (Local Intelligence and Independence) when an agent is isolated from other agents.
- If all of the connections among the agents are undamaged, through communication, the state estimations for all of the agents are consistent (Coordination and System Consistency).
- If some links between two agents are damaged, an agent can infer its local nodes based on its local measurements and the available messages received from its neighbors (Robustness and Optimization).

MSDBNs assume a static structure and a stable communication network for the whole system. However, for a real system, a sub Bayesian network node can be lost and the communication network can be shut down due to partial damage in the system. Therefore, on-line and automatic MSDBNs structure formation is necessary for making robust state estimations and increasing survivability of the whole system. Distributed Spanning Tree Optimization (DSTO), Distributed D-Sep Set Satisfaction (DDSSS), and Distributed Running Intersection Satisfaction (DRIS) combined with Distributed Belief Propagation (DBP) in MSDBNs makes inferences robust to partial damage in the whole system.

DSTO is a distributed spanning tree optimization algorithm for an undirected connected network with distinguished assigned weight of each edge. DSTO attains the optimal spanning tree with minimum summation of weights through at most $5N \log_2^{N+2E}$ message passing for an undirected graph with finite nodes N and edges E. Message transferring occurs only between neighboring nodes. Message passing and actions among different nodes are partially asynchronous and the whole process can be initiated at any node or any multiple nodes. DDSSS is a distributed d-sep set satisfaction algorithm for an undirected connected tree graph. DDSSS includes two processes: CollectPublicParInfo process followed by DistributePublicParInfo process. Through negotiations among neighboring nodes, shared nodes by two/multiple nodes are forced into d-sep sets. Similarly, message passing and actions among different nodes are partially asynchronous and the whole process can be initiated at any node. DRIS is a distributed running intersection property satisfaction algorithm for an undirected tree graph. DRIS only involves public information from the knowledge base of an individual Bayesian network. DRIS includes two processes also: CollectIndirectlyReachableAndPublicVars process followed by DistributeIndirectlyReachableAndPublicVars process. DBP is a distributed belief updating algorithm for a MSDBNs network. Through message passing between neighboring sub Bayesian networks, globally consistent inferences are reachable in each sub Bayesian network for its local nodes. Detailed descriptions and illustrative examples of DSTO, DDSSS, DRIS, and DBP were provided in Chapter IV.

Combining the distributed control architecture design and the distributed inference engine design together leads to a formal procedure of control system design for general large-scale complex systems. As applications of the proposed methodology, the control system designs of a simplified ship chilled water system and a notional ship chilled water system were demonstrated step by step. Simulation results not only show that the proposed methodology gives a clear guideline for control system design of general large-scale

complex systems with dynamic and uncertain environments, but also indicate that the combination of MSDBNs and HyMABC provides excellent performance for controlling general large-scale complex systems. Figure 211 shows a summary about how the proposed methodology and process are formed in this dissertation.

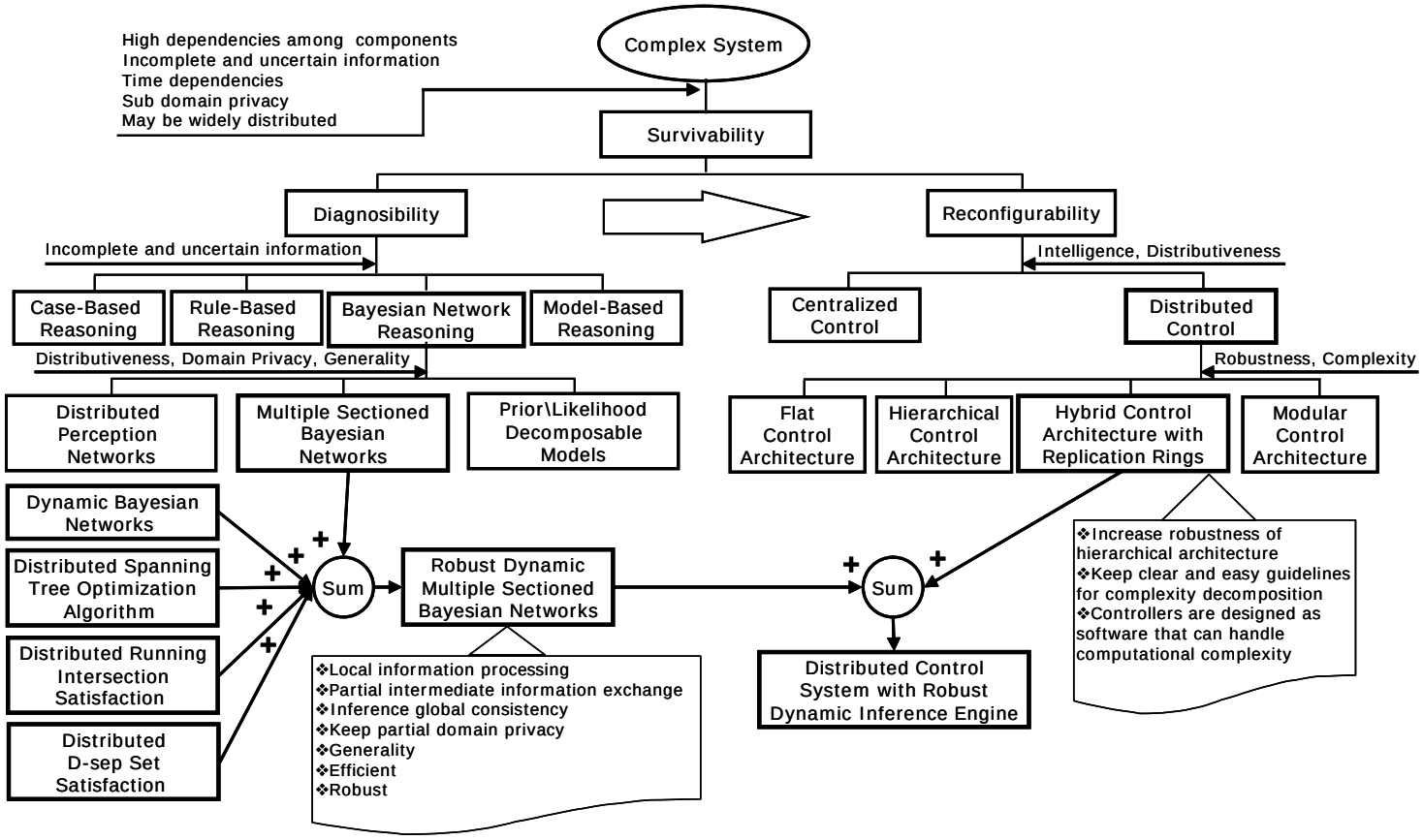


FIGURE 211 SUMMARIZATION OF THE RESEARCH TOPIC IN THIS DISSERTATION

7.3 Limitations on Application

The MSDBNs inference engine is suitable for a highly distributed non-recycled complex system with incomplete and uncertain information. It also has a series of limitations for some applications with the following characteristics, which should be addressed:

- Some directed cycles exist in the cause-effect relationships of a system and breaking the cycles will have significant impact on the modeling accuracies of the system dynamic characteristics. Bayesian networks cannot handle directed cycles.
- The structure and parameters of the cause-effect relationships vary frequently and dramatically. Efficient and effective on-line structure and parameters learning for Bayesian networks are needed.
- The system state variables are continuous with non-Gaussian distributions and need a large set of discrete states to represent. The computational complexity of a Bayesian network inference is exponentially proportional to the number of states of variables.
- Two subsystems share a relatively large set of variables compared to their domain set. Communication between the two subsystems will be heavy if a large set of variables are shared by them.
- A large number of variables are connected directly among different time steps in a dynamic system. More than two time-slices connected directly will cause the dynamic Bayesian network to be too complex to be used in practice efficiently.

7.4 Future work

The control system design for large-scale complex systems under significant uncertainties is itself a very complex and challenging problem. Although a seven-step process has been proposed and researched, this dissertation focused on step 3 to step 5, which mainly involved distributed control architecture design and distributed inference engine design. Other steps in the process are also very important and can be worked on as individual topics, such as system constraints and requirement analysis, complex system modeling and simulation [124, 125], complex system fault detection and isolation, agent internal logic designs and standardizations, etc.

In this dissertation, two-time slice homogenous DBN was used to model time-evolving events in a dynamic system. In some scenarios, more time slices may connect directly and the structure of the DBN may also be dynamic. DBN structure learning, adaptation, and efficient belief updating are on-going research fields [93, 126, 127]. The exact belief updating scheme used in this dissertation could be too slow for dense networks. Stochastic simulation as an approximating belief updating method may be more efficient. There are two ways for sampling in stochastic simulation: forward sampling and Gibbs sampling. Forward sampling, a random Monte Carlo Simulation, is very slow. Gibbs sampling is faster, but it needs to find a non-conflict instance of the whole network and cannot guarantee convergence. How to balance inference accuracy and time consumption by using stochastic modeling for specific systems is very challenging.

In this dissertation, sensor types, sensor quantities, and sensor locations were fixed. The inference engine used the available information to perform state estimations. In many practical systems, sensors are added in an ad hoc manner by the engineers using their experience in a qualitative way. Quantitative evaluation of sensor cost and performance will help design a sensor system that maintains a certain performance level at the lowest cost. A sensor system design includes many factors, such as sensor types, sensor quantities, sensor locations, sensor cost, inference accuracy, etc. There is no straightforward relation between the number of sensors and inference accuracy. The relevance of information gathered by different sensors has a significant impact on inference accuracy. In summary, a sensor network optimization for on-line automatic control of a large-scale complex system includes four aspects:

- Inference accuracy: determining state estimation accuracy of certain components, subsystems or the whole system.
- Inference efficiency: determining how fast the information can be handled to make inferences for certain components, subsystems or the whole system.

- Minimal sensor set: finding a minimal sensor set which achieves a specific degree of inference accuracy of certain components, subsystems or the whole system[128].
- Minimal cost sensors: in the case that different sensors are assigned with different costs, finding the minimal cost of a sensor set which can achieve a specific degree of inference accuracy of certain components, subsystems or the whole system[128].

For a distributed Bayesian network as described in this dissertation, data quality and availability can be simulated by observability of nodes. Through simulation and optimization methods, different sensor set and sensor deployments could be evaluated and an optimal one could be obtained.

Assume a distributed Bayesian network with n nodes $V = U \cup O$, where U is the set of unobservable nodes and O is the set of possible observable nodes under current techniques and situations; $O = O_s \cup O_u$, $O_s \cap O_u = \varnothing$, where O_s is the set of variables actually measured by sensors and O_u is the set of non-measured variables; $U = U_q \cup U_n$, $U_q \cap U_n = \varnothing$, where U_q is the critical query set and U_n is the insignificant query set .

$$\underset{O_s \subseteq O}{Minimize} \quad J = w_1 E\left(C\left(O_s\right)\right) + w_2 E\left(T\left(O_s\right)\right) + w_3 \frac{1}{E\left(A\left(U_q, O_s\right)\right)} \tag{7.1}$$

Where w_1, w_2, w_3 are weighting coefficients, $E\left(C\left(O_s\right)\right)$ is the cost expectation of the set of sensors measuring O_s; $E\left(T\left(O_s\right)\right)$ is the expected information process time from the set of O_s and $E\left(A\left(U_q, O_s\right)\right)$ is the expectation of inference accuracy of the critical query set U_q based on the information from O_s .

As described in the previous sections, Bayesian networks have the capability of handling incomplete data sets, so all of the algorithms of MSDBNs proposed in this dissertation for different scenarios will be the same and the only difference is the objective function J. It is a typical discrete variable optimization problem.

The minimization problem can be transformed into other forms as follows:

$$\begin{aligned} &\underset{O_s \subseteq O}{Minimize} \quad J = \frac{1}{E\left(A\left(U_q, O_s\right)\right)} \\ &subject\ to \\ &\qquad E\left(C\left(O_s\right)\right) \le C_{UpperLimit} \\ &\qquad E\left(T\left(O_s\right)\right) \le T_{UpperLimit} \end{aligned} \tag{7.2}$$

The exact belief updating scheme of MSDBNs is for a system with discrete state variables. This scheme can be extended to handle linear dynamic continuous system state estimations with Gaussian distributions, or a hybrid system with both continuous variables and discrete variables. Furthermore, for nonlinear systems with Gaussian noises, linearization of the system dynamic can be used to approximate inferences. However, for a nonlinear system with non-Gaussian noises and continuous hidden variables, exact inference is not possible [93] .

The control system proposed in this dissertation is an Integrated, Reconfigurable, and Intelligent System (IRIS). It provides the capability of on-line automatic data collection, data processing, data transferring, fault detection, fault identification, fault isolation, and system reconfiguration. However, for a real system with time constraints, algorithms of information abstraction, processing, and communication should be numerically efficient. Although a framework of IRIS design is given in this dissertation, clearly, IRIS design itself is a very challenging control design task and consists of many advanced and interesting topics for on-going or future research, such as efficient information

compression, complete asynchronous message passing with approximate inferences, more efficient belief updating schemes for a dynamic system, tradeoffs among cost, time, inference accuracy, control effectiveness, etc.

APPENDIX A

FULLBNT

FullBNT is an open source Bayes Network Toolbox written in MATLAB script under the terms of the GNU Library General Public License. It was firstly developed by Kevin P Murphy in 1999. It is a comprehensive MATLAB toolbox for individual probabilistic network modeling and reasoning. The latest version is FullBNT-1.0.4 and it is continuously updated by Kevin P Murphy and his students. FullBNT-1.0.4 consists of several sub toolboxes, such as Bayes Net Toolbox (BNT), graph, GraphViz, HMM, Kalman, etc. Detailed introduction can be found on Kevin P Murphy personal homepage [129]. In this dissertation, BNT is the toolbox heavily used for Bayesian network inferences. However, BNT is developed for individual Bayesian network inferences. In order to make it suitable for MSDBNs inferences, a few modifications/added functions need to be done based on the original code. The following context lists a set of the modifications/added functions.

```
-----------------------------------------------------------------------------------------------------------
function [orderedNodesID, orderedNodesSize,
orderedNodesDistribution]=getOrderedNodesDistribution(nodesID,nodesDistribution)
%Reorder the distribution table according to the nodesID from small to big.
if(isvector(nodesDistribution))
    orderedNodesID=nodesID;
    orderedNodesSize=length(nodesDistribution);
    orderedNodesDistribution=nodesDistribution;
    return
end

orderedNodesID=sort(nodesID);
if(orderedNodesID==nodesID)
    orderedNodesID=nodesID;
    orderedNodesSize=size(nodesDistribution);
    orderedNodesDistribution=nodesDistribution;
    return
end

nodesNum=length(nodesID);
nodesPositionIndex=zeros(1,nodesNum);
```

```
for(i=1:nodesNum)
    nodesPositionIndex(i)=find(nodesID==orderedNodesID(i));
end

orderedNodesDistribution=permute(nodesDistribution,nodesPositionIndex);
orderedNodesSize=size(orderedNodesDistribution);
%END
-------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
function
[hostCliqueID,linkages,peerSepset,linkageSeparator]=get_linkagetree(cliques,separator,sha
redVars)
%Get linkage tree from the set of cliques of original Bayesian network for%shared
variables with its neighbors
num=length(cliques);
for i=1:num
    cliquesI=cliques  work;
    cliquesITemp=[];
    count=0;
    for j=1:length(cliquesI)
        if(~(length(find(sharedVars==cliquesI(j)))==0))
            count=count+1;
            cliquesITemp(count)=cliquesI(j);
        end
    end
    cliques{i}=cliquesITemp;
end

for i=1:num
    cliquesI=cliques{i};
    if(length(cliquesI)==0)
        continue
    end

    for j=i+1:num
        cliquesJ=cliques{j};
        if(length(cliquesJ)==0)
            continue
        end
        temp=intersect(cliquesI,cliquesJ);
        if(length(temp)==length(cliquesJ))
            cliques{j}=[];
        else if(length(temp)==length(cliquesI))
                cliquesI=[];
                cliques{i}=[];
                break
            end
        end
    end
end

for i=1:num
    cliquesI=cliques{i};
    cliques{i}=[i,cliquesI];
end

count1=-1;
count2=0;
for i=1:num
    cliquesI=cliques{i};
    if(length(cliquesI)==1)
        count1=count1+1;
        separator(i-count1,:)=[];
        separator(:,i-count1)=[];
    else
      count2=count2+1;
      tempCliques,=cliquesI;
    end
end
```

```
cliques=tempCliques;
num=length(cliques);
for i=1:num
    for k=i+1:num
    separatorIK=separator{i,k};
    separatorIKTemp=[];
    count=0;
    for j=1:length(separatorIK)
        if(~(length(find(sharedVars==separatorIK(j)))==0))
            count=count+1;
            separatorIKTemp(count)=separatorIK(j);
        end
    end
    separator{i,k}=separatorIKTemp;
    separator{k,i}=separatorIKTemp;
    end
end

for i=1:num
    hostCliqueID(i)=cliques{i}(1);
    cliques{i}(1)=[];
end

linkages=cliques;
linkageSeparator=separator;
peerSepset=linkages;
peerSepset{1}=[];

for i=2:num
    for j=1:i-1
        if(length(linkageSeparator{i,j})>0)
            peerSepset{i}=linkageSeparator{i,j};
            break
        end
    end
end
%END
```

```
function [clpot, seppot] = collect_clique_evidence(engine, clpot, seppot)
% COLLECT_EVIDENCE Do message passing from leaves to root (children then parents)
% [clpot, seppot] = collect_evidence(engine, clpot, seppot)

for n=engine.postorder %postorder(1:end-1)
    for p=engine.postorder_parents{n}
        clpot{p} = divide_by_pot(clpot{p}, seppot{p,n}); % dividing by 1 is redundant
        seppot{p,n} = marginalize_pot(clpot{n}, engine.separator{p,n}, engine.maximize);
        clpot{p} = multiply_by_pot(clpot{p}, seppot{p,n});
    end
end
%END
```

```
function linkageI_pot=get_linkageI_pot(linkageIDomain,linkageISizes,linkageIDistVec)
linkageIDist=vectorToMultiDim(linkageIDistVec,linkageISizes);
[orderedlinkageIDomain, linkageISizes, linkageIDist] =
getOrderedNodesDistribution(linkageIDomain,linkageIDist);
linkageI_pot=dpot(orderedlinkageIDomain, linkageISizes, linkageIDist);
%END
```

```
function linkageIndex=getLinkageIndex(linkageI,linkages)
len=length(linkages);
linkageIndex=0;
for i=1:len
    temp=linkages{i};
    intersectI= intersect(temp,linkageI);
    lenInterI=length(intersectI);
```

```
    if((lenInterI==length(temp))&&(lenInterI==length(linkageI)))
        linkageIndex=i;
        return
    end
end

if(linkageIndex==0)
    error('[no matched linkage in linkages]');
end
%END
```

```
function b=vectorToMultiDim(a,sizeB)
%Transform a vector to a multi-dimension matrix. The matrix dimension is
%less 9
if(length(sizeB)==1)
    b=a;
    return
end

  numInA=length(a);
  numInB=1;
for(i=1:length(sizeB))
    numInB=numInB*sizeB(i);
end

if(numInA~=numInB)
    error('[the elment number in input vector is not equal to elment number in the
required result matrix]');
    return
end


count=0;
dimB=length(sizeB);

if(dimB==2)
    for(j=1:sizeB(2))
        for(i=1:sizeB(1))
            count=count+1;
            b(i,j)=a(count);
        end
    end
    return;
end

if(dimB==3)
    for(k=1:sizeB(3))
        for(j=1:sizeB(2))
            for(i=1:sizeB(1))
                count=count+1;
               b(i,j,k)=a(count);
            end
        end
    end
    return;
end


if(dimB==4)
    for(l=1:sizeB(4))
        for(k=1:sizeB(3))
            for(j=1:sizeB(2))
                for(i=1:sizeB(1))
                    count=count+1;
                   b(i,j,k,l)= a(count);
                end
            end
        end
    end
```

```
    return;
end



if(dimB==5)
    for(m=1:sizeB(5))
        for(l=1:sizeB(4))
            for(k=1:sizeB(3))
                for(j=1:sizeB(2))
                    for(i=1:sizeB(1))
                        count=count+1;
                       b(i,j,k,l,m)=a(count);
                    end
                end
            end
        end
    end
    return;
end

if(dimB==6)
    for(n=1:sizeB(6))
        for(m=1:sizeB(5))
            for(l=1:sizeB(4))
                for(k=1:sizeB(3))
                    for(j=1:sizeB(2))
                        for(i=1:sizeB(1))
                            count=count+1;
                            b(i,j,k,l,m,n)=a(count);
                        end
                    end
                end
            end
        end
    end
    return;
end


if(dimB==7)
    for(p=1:sizeB(7))
        for(n=1:sizeB(6))
            for(m=1:sizeB(5))
                for(l=1:sizeB(4))
                    for(k=1:sizeB(3))
                        for(j=1:sizeB(2))
                            for(i=1:sizeB(1))
                                count=count+1;
                                b(i,j,k,l,m,n,p)=a(count);
                            end
                        end
                    end
                end
            end
        end
    end
    return;
end

if(dimB==8)
    for(r=1:sizeB(8))
        for(p=1:sizeB(7))
            for(n=1:sizeB(6))
                for(m=1:sizeB(5))
                    for(l=1:sizeB(4))
                        for(k=1:sizeB(3))
                            for(j=1:sizeB(2))
                                for(i=1:sizeB(1))
                                    count=count+1;
                                    b(i,j,k,l,m,n,p,r)=a(count);
```

```
                                end
                            end
                        end
                    end
                end
            end
        end
    end
    return;
end

if(dimB>8)
    error('[multiDimToVector function can not handle that matrix dimension >8. ]');
end
%END
```

APPENDIX B

JMATLINK INTRODUCTION AND MANUAL UNDER WINDOWS

JMatlink is an open source Java package under the terms of the GNU Library General Public License originally developed by Stefan Müller in 1999, which connects Java to MATLAB using Java Native Interface (JNI). The latest version is JMatLink V1.3.0 released in December, 2005. Both of JMatLink.jar file and the resource code can be downloaded from http://www.held-mueller.de/JMatLink. JMatLink supports multiple MATLAB instances open from Java for individual MATLAB installation. Another similar tool is JLab and it was developed by Iain E. Toft in January 2005 [130]. However, JLab only allows opening single MATLAB instance for individual MATLAB installation and not suitable for the purpose in this dissertation. JMatLink provides a convenient way to combine the powerful computational engine of MATLAB, real-time hardware, and the power of web-based and GUI functions of JAVA[131].

The core part of JMatlink includes three JAVA classes and one C file. The three JAVA classes are CoreJMatLink.class, JMatLink.class and JMatLinkException.class. CoreJMatLink.class is the core class of JMatLink. It makes connection to the "C" file through JNI. The C file is compiled into a DLL file, which actually calls various C functions existing in MATLAB engine. JMatlink can be compiled under both of Windows operation systems and Unix operation systems. In this dissertation, JMatLink is used under Windows Vista. The following section will give a simple introduction about how to use JMatLink under Windows Vista step by step.

- Step 1: Set up system environment variable.

Right click Computer > Properties >Advanced System Settings, system property window pops up as shown in Figure 212.

Click tab Advanced > Environment Variables, environment variables window pops up as shown in Figure 213.

Double click PATH. Add the JAVA run environment bin directory of JAVA installation directory\SDK\jdk\bin and MATLAB bin directory of MATLAB installation directory\matlab\bin to the existed value of PATH.

Double click CLASSPATH and add …\JMatLink.jar to the existed value of CLASSPATH.

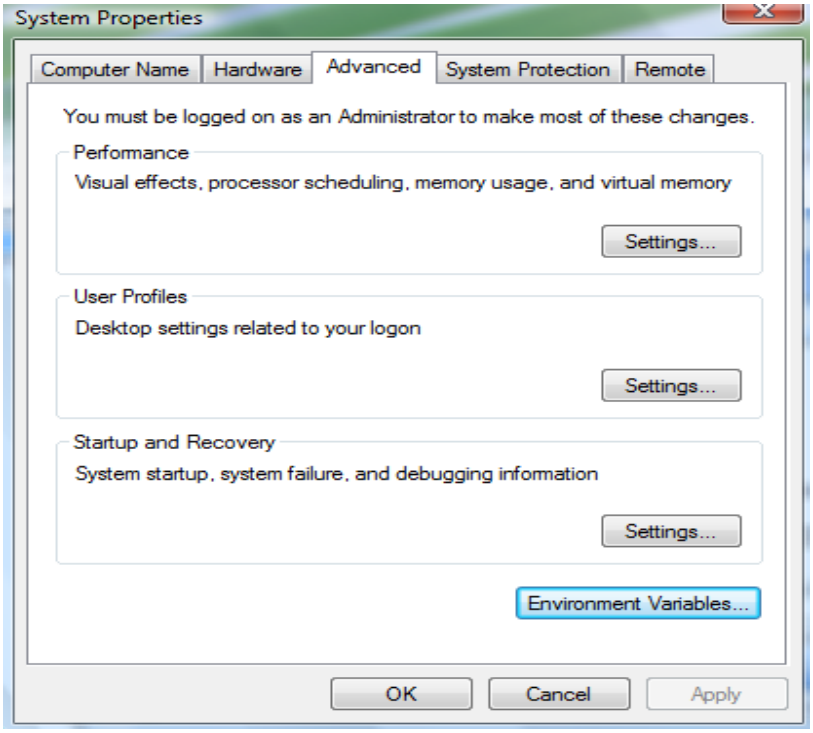


FIGURE 212 SYSTEM PROPERTIES WINDOW

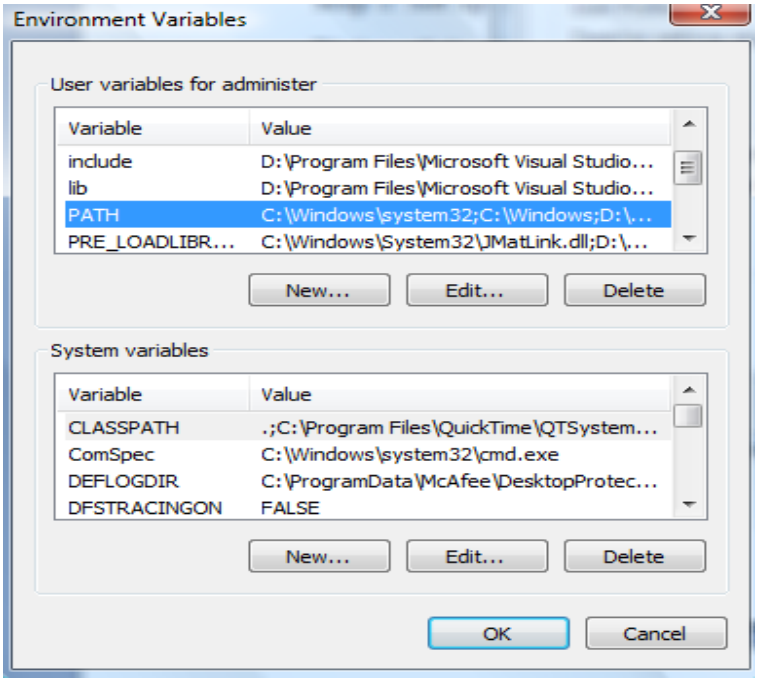


FIGURE 213 ENVIRONMENT VARIABLES WINDOW

- Step 2: Copy the JMatLink.dll into the Windows/system32 directory.

If JMatLink.dll does not match the current running operation system, it can be recompiled by using some Integrated Development Environments (IDEs) for C/C++, such as Visual C++, MinGW in Eclipse etc. Detailed instructions of how to create DLL file can be found in many JNI implementation documents on line.

- Step 3: Import the JMatLink.class into the customer defined classes and use it as a regular JAVA class as the example code shown below.

If JMatLink.class does not match current JAVA version due to some deprecated methods/classes, it can be recompiled and recompressed by using any standard JAVA compilers. Normally, it is not necessary to recompile the class files, because JAVA new versions are always backward compatible.

Note: JMatLink allows opening any number less than 10 MATLAB instances from JAVA for any single standalone MATLAB installation. However, if multiple MATLAB instances are open from different threads, memory violations will show up randomly. It is a deeply hidden bug and the debugging error messages are general, misleading and non-reduplicable. Make sure that multiple instances are initiated from the same thread for one single MATLAB installation. Figure 214 shows a simple example of using JMatLink in Java.

```
//import JMatLink.class
import jmatlink.JMatLink;

public class TestJMatLink {
    private JMatLink engineJMatLink;

    public void test(){
        //Create a JMatLink object
        engineJMatLink = new JMatLink();
        long a;
        //Open a single instance of MATLAB Engine
        a=engineJMatLink.engOpenSingleUse();

        double[][] testArray={{1,2},{3,4}};
        //Put a matrix from JAVA to MATLAB workspace
        engineJMatLink.engPutArray(a,"testArray", testArray);

        //Read a matrix from MATLAB workspace to JAVA
        double[][] testArrayFromMATLAB=engineJMatLink.engGetArray(a,"testArray");
        System.out.println(testArray[0][0]+", "+testArray[0][1]+", "+testArray[1][0]+", "+testArray[1][1]);
    }

    public  static void main(String[] args){
        TestJMatLink instanceOfTestJMatLink=new TestJMatLink();
        instanceOfTestJMatLink.test();
    }
}
```

FIGURE 214 EXAMPLE CODE OF USING JMATLINK IN JAVA

APPENDIX C

IMPORTANT STEPS OF JAVA PLUG-INS IN MODELCENTER

ModelCenter supports Java plug-ins through Analysis Server. Herein, the main steps about how to implement a Java plug-in in ModelCenter are listed as follows:

- Add "permission java.security.AllPermission;" to aserver.policy which is located in the directory Phoenix Integration\Analysis Server 5.1. This step is to allow some popup GUIs shown in ModelCenter; otherwise, ModelCenter will give some security exceptions if a plug-in tries to show some GUIs.
- Put aserver.zip in the global class path. This step is to make Analysis Server work for the plug-ins.
- Put all of the jade *.jar files and other *.jar, *.zip, *.rar, etc. files used in the applications into the directory of Phoenix Integration\Analysis Server 4.1\jre\lib\ext. Analysis Server uses its own jre. It does not care if the jade *.jar files are put into the global class path or not because it does not search these class paths. Analysis Server only searches the *.jar files in its own jre directory. In the application of this dissertation, two other library files: Jama-1.0.1.zip and JMatlink.jar are used. Jama-1.0.1.zip consists some classes which make the manipulation of matrix much easier. It is developed by National Institute Standards and Technology. JMatlink is an open source Java package under the terms of the GNU Library General Public License originally developed by Stefan Müller in 1999, which connects Java to MATLAB by using Java Native Interface (JNI) as described in Appendix B.

- Separate the component classes file with some other classes files which the components will use. Put all of the components *.jar files into the directory of Phoenix Integration\Analysis Server4.1\analysises while put all of the other classes *.jar files into the directory of Phoenix Integration\Analysis Server 4.1\jre\lib\ext.
- Now launch ModelCenter and Analysis Server. Double click aserv://localhost, the Java plug-in will be shown in the left bottom of ModelCenter window. Drag the plug-in into the analysis view window and type a name for this component.

The following is a simple example to indicate how to create *.jar files for a java component in Model Center.

First, create a class file which implements IPHXAnalysis interface. In this class, inputs and outputs for the plug-in need to be specified. For inputs, implement getXXX( ) and setXXX( ) methods while for outputs, implement getXXX( ) methods. Execute( ) method is called when this java Plug-in Contribution Analysis (CA) is scheduled to run in ModelCenter. Therefore, implementing the method execute( ) is the core part of the application.

```
-----------------------------------------------------------------------------------------------------
//A example class resource code of Java Plug-in in ModelCenter

package javaPlugin;

import com.phoenix_int.aserver.*;

public class JavaPlugin implements IPHXAnalysis{
	private double inputX;
	private static double outputY;

	public JavaPlugin(){
		System.out.println("In JavaPlugin Constructor:");
		inputX=0;
		outputY=1;
	}

	public double getInputX() {
		return inputX;
	}

	public void setInputX(double X) {
		this.inputX = X;
	}

	public double getOutputY() {
```

```
                return outputY;
        }

        //execute()function will be called when ModelCenter schedules this Java Plugin
Contribute Analysis(CA) to run
        public void execute() throws Exception {
                outputY=inputX*inputX;
        }

        public void end() throws Exception {

        }

/*In order to include standard Analysis Server descriptive fields, the user must
implement specific static methods in the class that return a string.
*/
   public static String getAuthor(){
           return "Daili Zhang";
   }
   public static String getDescription(){
           return "This is an example of how to use Java Plugin in ModelCenter";
   }
   public static String getHelpURL(){
           return "Null";
   }
   public static String etIconFile(){
           return "No iconfile specified in this example";
   }
   public static String getKeywords(){
           return "example, Java Plugin, ModelCenter";
   }
}
```

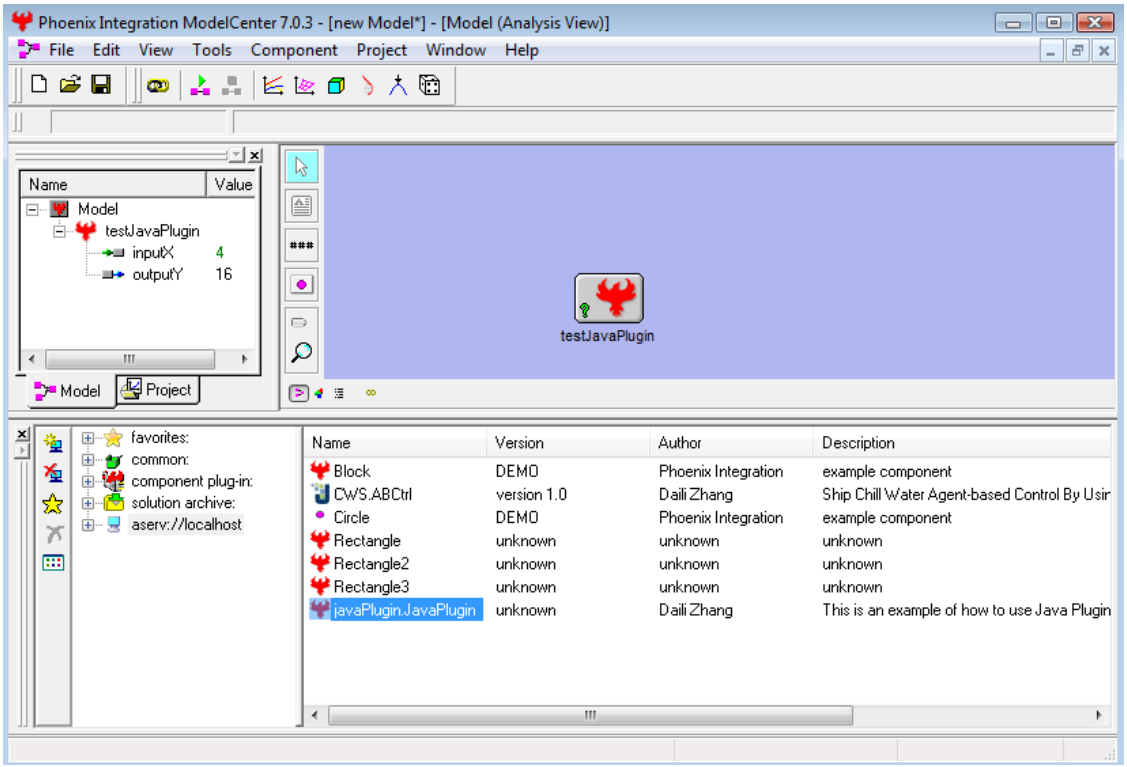


FIGURE 215 A JAVA PLUG-IN EXMAPLE IN MODELCENTER

Second, create a *.stub manifest file based on the established Java plug-in class. For example, a manifest file JavaPlugin.stub is created in Notepad and its content is as follows:

```
---------------------------------------------------------------------------------------------------------

Name: javaPlugin/JavaPlugin.class
Java-Bean: True

---------------------------------------------------------------------------------------------------------
```

Note that a blank line must separate each entry in the manifest.stub file, and the file must begin and end with a blank line.

Last, compress the class file into a *.jar file by using jar command with specific parameters. The commands for this example are shown in the following:

```
---------------------------------------------------------------------------------------------------------
cd D:\Data\AboutThesis\MABCtrlDesginForRSAD\MABCtrlDesignForRSAD\bin
jar -cfm javaPlugin.jar javaPlugin.stub javaPlugin/
---------------------------------------------------------------------------------------------------------
```

Once the javaPlugin.jar file is created, process step 1 to step 5, and then the results are shown in Figure 215.

APPENDIX D

VISUAL BASIC PLUG-INS CODE IN MODELCENTER

The Visual Basic plug-in in the application of this dissertation is used to integrate Flowmaster model into ModelCenter. The source code shown in the following is modified from the source code written by my colleague Kyungjin Moon. More detailed information about how to implement Visual Basic plug-ins in ModelCenter is available in a developer's guide: Plug-Ins for ModelCenter® [132].

```
---------------------------------------------------------------------------------------------------------
    Option Explicit
    Implements IComponentPlugIn

    'modelcenter objects
    Dim mc As modelCenter.Application
    Dim atm As IAddToModel

    'flowmaster objects
    Dim Analysis As FM2_AnalysisControl

    Dim comp As IComponent
    Dim vars As IVariables
    Dim t0 As Double
    Dim deltaT As Double
    Dim timeStep

    'internal variables and objects
    Dim i As Integer
    Dim intResultFileID As Integer
    Dim numOfRunSteps As Integer
    Dim indexOfControllers(120) As Integer
    Dim loginOk As Boolean
    Dim sInitialFile As String

    Dim iniData As Workbook
    Dim nameController(1 To 100, 1 To 150) As String
    Dim nameGauge(1 To 100, 1 To 150) As String
    Dim indexController(1 To 100) As Integer
    Dim indexGauge(1 To 100) As Integer
    Dim numControllers As Integer
    Dim numGauges As Integer
    Dim nameGroupControllers(1 To 100) As String
    Dim nameGroupGauges(1 To 100) As String

    Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long, ByVal
hWndNewParent As Long) As Long

    Private Sub IComponentPlugIn_construct(ByVal modelCenter As Object, ByVal addToModel
As Object, Optional ByVal dataCollector As Object = Nothing)
       Set mc = modelCenter
```

```
   Set atm = addToModel

End Sub

Private Sub IComponentPlugIn_fromString(ByVal setupString As String)
End Sub

Private Sub IComponentPlugIn_onEnd()
End Sub

Private Sub IComponentPlugIn_run()

    Set comp = atm.getComponent()
    Set vars = comp.Variables

    t0 = vars.Item("t0").Value
    deltaT = vars.Item("deltaT").Value

    vars.Item("timeStep").Value = Analysis.deltaT

    numOfRunSteps = Round(deltaT / Analysis.deltaT, 0)

    ' reading inputs from ModelCenter and assigning it ot flowmaster controllers
    Dim i As Integer, j As Integer
    Dim aCtrler As IFM2_Controller
    Dim aGauge As IFM2_Gauge

    Dim aInput As IDoubleArray

    For i = 1 To numControllers
       Set aInput = vars.Item(nameGroupControllers(i))
       For j = 1 To indexController(i)
           Set aCtrler = Analysis.GetControllerBN(nameController(i, j))
           aCtrler.OutputValue = aInput.Value(j - 1)
       Next
    Next

    'Run
    For i = 1 To numOfRunSteps
    Analysis.RunToNextTimeStep
    Next

    'Send the current simulation time
    vars.Item("t").Value = vars.Item("t").Value + deltaT

    'reading outputs from flowmaster and showing it in modelcenter
    For i = 1 To numGauges
        Set aInput = vars.Item(nameGroupGauges(i))
        For j = 1 To indexGauge(i)
            Set aGauge = Analysis.GetGaugeBN(nameGauge(i, j))
            aInput.Value(j - 1) = aGauge.OutputValue
       Next
    Next

End Sub

'Public Sub Analysis_OnNewTimestep(ByVal StepNo As Long, ByVal Time As Long)
'    Set currTime = vars.Item("Time")
'    currTime.Value = Time
'End Sub

Private Function IComponentPlugIn_show() As Boolean

    ' Open the Initial data set for RSAD.xls
    sInitialFile = "D:\AboutThesis\FlowMasterModelV1\fluid_network_v2.xls"
    Set iniData = Workbooks.Open(FileName:=sInitialFile)

    ' Initial setting of time
    atm.addInput "t0", "double", "0"
    atm.addInput "deltaT", "double", "0.1"
    atm.addOutput "t", "double", "0"
```

```
        atm.addOutput "timeStep", "double", "0.05"

       ' Initialization of FlowMaster analysis
       Set Analysis = New FM2_AnalysisControl
       'set the data manager directory
       loginOk = Analysis.DatabaseLogin("localhost\SQLExpress", "Flowmaster",
"Flowmaster", "Admin", "")
       'Open the project
       Analysis.ProjectName = "flowmaster"
       'Open the network
        Analysis.NetworkName = "fluid_network_v2"

        Analysis.CallerVersion = "VB"
        Analysis.ResultFileID = 1
        Analysis.AnalysisType = "st"
        Analysis.HeatTransfer = 1
        Analysis.Initialise

    '------------------------ ESTABLISH INPUTS & OUTPUTS--------------------------

    numControllers = 0
    numGauges = 0

    Dim NameBuffer As String, iniStates As String
    Dim i As Integer, j As Integer, counterController As Integer, counterGauge As Integer

    i = 2
    counterController = 1
    counterGauge = 1
    While iniData.Worksheets(1).Cells(1, i).Value > 0
        If iniData.Worksheets(1).Cells(1, i).Value = "Controller" Then
            iniStates = iniData.Worksheets(1).Cells(3, i + 1).Value
            nameController(counterController, 1) = iniData.Worksheets(1).Cells(3,
i).Value
            j = 4
            While iniData.Worksheets(1).Cells(j, i).Value > 0
                iniStates = iniStates & ";" & iniData.Worksheets(1).Cells(j, i + 1).Value
                nameController(counterController, j - 2) = iniData.Worksheets(1).Cells(j,
i).Value
                j = j + 1
            Wend
            indexController(counterController) = j - 3
            numControllers = numControllers + 1
            NameBuffer = iniData.Worksheets(1).Cells(2, i).Value
            nameGroupControllers(counterController) = NameBuffer
            atm.addInput NameBuffer, "double[]", iniStates
            counterController = counterController + 1
        ElseIf iniData.Worksheets(1).Cells(1, i).Value = "Gauge" Then
            iniStates = iniData.Worksheets(1).Cells(3, i + 1).Value
            nameGauge(counterGauge, 1) = iniData.Worksheets(1).Cells(3, i).Value
            j = 4
            While iniData.Worksheets(1).Cells(j, i).Value > 0
                iniStates = iniStates & ";" & iniData.Worksheets(1).Cells(j, i + 1).Value
                nameGauge(counterGauge, j - 2) = iniData.Worksheets(1).Cells(j, i).Value
                j = j + 1
            Wend
            indexGauge(counterGauge) = j - 3
            numGauges = numGauges + 1
            NameBuffer = iniData.Worksheets(1).Cells(2, i).Value
            nameGroupGauges(counterGauge) = NameBuffer
            atm.addOutput NameBuffer, "double[]", iniStates
            counterGauge = counterGauge + 1
        End If
        i = i + 4

    Wend

    ' Current time in seconds
        atm.updateComponent

        IComponentPlugIn_show = False
```

```
        iniData.Close SaveChanges:=False
    End Function

    Private Function IComponentPlugIn_toString() As String
       IComponentPlugIn_toString = ""
    End Function
```

APPENDIX E

JAVA SOURCE CODE

All agents established for the control system of the application in this dissertation are based on JADE (Java Agent DEvelopment Framework). JADE is an open source software framework developed and distributed by Telecom Italia. More functions and capabilities are added to JADE continuously. The current version is JADE 3.7 and was released in February, 2009. The core part of JADE includes 5 *.jar files: commons-codec-1.3.jar, http.jar, iiop.jar, jade.jar, jadeTools.jar, which can be downloaded from JADE website [133]. JADE also supports many add-ons for function extensions. More information about JADE is available on JADE website [133]. In the application of this dissertation, there are 30 classes which are distributed into 9 packages: bayesianNetworkAgent, chilledwaterresource, component, crossvalves, CWS, interfacesWithOtherModules, msbn.unil, RSADsystem, serviceload. Since the space is limited, only several main classes' source code is provided in the following.

```
--------------------------------------------------------------------------------------------------------
package bayesianNetworkAgent;

import msbn.util.*;

import java.io.IOException;
import java.util.*;

import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;

import jmatlink.*;

/**************************************************************************** * This
agent has two inputs:
 * 1. the information about the Bayesian network
```

```
 * BN Alias Name(String); nodes name(String[]);
 * nodesSize(Map(nodeName,nodeSize));
 * intrac(String[][2], like {"node1","node2";"node2","node3"} means node1 is parent of
node2 and node2 is parent of node3;
 * neighborsInfo(Map(neighborName,sharedVariable(Set))
 * 2. the directory where FullBNT1.0.4 installed
 * @author Daili Zhang
 * @author Aerospace System Design Lab, Georgia Institute of Technology.
 * @version 1.0
 */

public class BayesianNetworkAgent extends Agent{
	public static JMatLink engineJMatLink=new JMatLink(); //Establish a link between
this BN network agent with a Matlab instance.
	private long EPI; //EPI for a single use of Matlab

	private String BNName; //Agent name
	private String[] nodesName; //BN network nodes' names, after setup(), the
nodesName are in order from parents to children
	private double[]  nodesSize; //Each node's size. It has been in order after
setup().
	private String[][] intrac; //The causal relationships. every row indicates one
causal relationship, such as {node1Name, node3Name} means node1 is the parent of node3
	private Map nodesSizeMap=new HashMap(); //(key=nodesName;value=nodesSize)
	private Map neighborsInfo=new HashMap(); //Map(key=neighbors name, value=shared
variables' names)
	private Map neighborsInfoID=new HashMap(); //Map(key=neighbors name, value=shared
variables' IDs (numbers))
	private Map conProbDis=new HashMap(); //This table stores information about the
conditional probability for each nodes(key="nodeName",value=(Map)tableMap), and
tableMap=(key=String[]nodesName,key=double[][][]...(conditional prob table))
	private Map orderedConProbDis=new HashMap(); //This table is similar to conProDis,
but the conditional prob table's domain are in order.

	private Map componentSt=new HashMap(); //(key=componentName(component agent
name),value=BNNetwork Node Name;
	private Map componentSpt1=new HashMap(); //(key=componentName,value=BNNetwork Node
Name;)
	private Map componentCpt1=new HashMap(); //(key=componentName,value=BNNetwork Node
Name;)
	private Map observationOt=new HashMap(); //(key=sensorName(sensor agent
name),value=BNNetwork Node Name;)
	private Map componentODt=new HashMap(); //(key=componentName,value=BNNetwork Node
Name;)

	private Map hardEvidence=new HashMap(); //(key=node name; value=evidence(double))
	private Map softEvidence=new HashMap(); //(key=node name;
value=evidence(double[]));

	private Map linkagesEvidence=new HashMap(); //(key=neighbor
name;value=(Set)linkagesInfoFromNeighbors);an element in the set is an object from Class:
TransferedMessagesForCoBeliefUpdate.

	private Map nodesNamePosition=new HashMap(); //every node name with its ID in the
network. The ID is a number from 1 to N
	private Map nodesPositionName=new HashMap(); //if a node ID is known, trace this
node's name according to its ID.
	private double[][]  BNDAG; //BN network directed acyclic graph in a matrix form.
	private double[][]  BNMod; //BN network indirected graph in a matrix form after
moralization.
	private double[][]  BNTri; //BN network indirected graph in a matrix form after
triangulation.
	private double[] eliminateOrder; //After triangulation, what is the best
elimination order.
	private Map cliques=new HashMap(); //(key=cliqueID(number);value=(double[])the
clique's domain.

	private boolean getMessageFromASystem=false; //If it is true, this agent will
initiate coTriangulation and coBeliefupdating
	private int countReceivedMesOfFillInsFeedBack=0; //How many FillInsFeedBack this
BN network has already received
```

```
        private int countReceivedUpdateBeliefMessages=0; //How many UpdateBeliefMessages
this BN network has already received
        private String recordDFESender; //record which BN network sends
DepthFirstEliminate operation to this BN network. FillInsFeedBack message from this BN
network will be sent to DFEsender.
        private String recordCollectBeliefSender; //record which BN network sends
CollectBelief operation to this BN network. UpdateBelief message from this BN network
will be sent to CollectBeliefSender.

        private Map componentStDist=new HashMap();

        private String existenceOfFillIns="0"; //after one cotriangulation cycle, is there
any fill ins introduced in this network? If it is, "1", else "0";
        private String LJFBUSysAgentName=ConstantCollection.LJFBUSysAgentName;
        private AID LJFBUSysAgent;

        //check component name position,why sometimes it is out of the index.


        MessageTemplate mtForCoTriangulation;
        MessageTemplate mtForCoBeliefUpdate;
        MessageTemplate mtForTriagulationOver;
        MessageTemplate mtForComponent;
        MessageTemplate mtForSensor;

        public void setup(){
                BNName=getAID().getLocalName();

                //arguments have two parts: arguments[0] is an BNInfo class instance;
arguments[1] is the path where Matlab should be set to in order to use FullBNT functions.
                Object[] arguments=getArguments();
                if(arguments!=null){
                        LJFBUSysAgent=new AID(LJFBUSysAgentName,AID.ISLOCALNAME);
                        BNInfo bnInfoTemp;
                        bnInfoTemp=(BNInfo)arguments[0];

                        //The path where FullBNT1.0.4 installed
                        String path=(String)arguments[1];

                        //extract all information about this BN network.
                        neighborsInfo=bnInfoTemp.getNeighborsInfo();
                        nodesName=bnInfoTemp.getNodesName();
                        nodesSizeMap=bnInfoTemp.getNodesSize();
                        intrac=bnInfoTemp.getIntrac();
                        conProbDis=bnInfoTemp.getConProbDis();

                        componentSt=bnInfoTemp.getComponentSt();
                        componentSpt1=bnInfoTemp.getComponentSpt();
                        componentCpt1=bnInfoTemp.getComponentCpt();
                        observationOt=bnInfoTemp.getObservationOt();
                        componentODt=bnInfoTemp.getComponentODt();

                        //get an JMatLink engineJMatLink and make it work for FullBNT 1.0.4
                        //engineJMatLink = new JMatLink();
                        EPI=engineJMatLink.engOpenSingleUse();

                        engineJMatLink.engEvalString(EPI,"cd('"+path+"')");
                        engineJMatLink.engEvalString(EPI,"addpath(genpathKPM(pwd))");

                        engineJMatLink.engSetVisible(EPI,true);
                        engineJMatLink.setDebug(false);
                        engineJMatLink.engEvalString(EPI,"clc");
                        engineJMatLink.engEvalString(EPI,"clear all");

                        //change the workspace directory
                        //String
workspaceDir="D:\\AboutThesis\\BayesianNetworksInMatlab\\"+BNName;
                        //engineJMatLink.engEvalString(EPI,"cd('"+workspaceDir+"')");
                        //engineJMatLink.engEvalString(EPI,"path(path,'"+path+"')");
```

```
                        //establish the Bayesian network in the Matlab workspace.
                        //[intra,names] are stored in the Matlab workspace.
                        Map temp1=new HashMap();
                        temp1=CallMatlabFunsAndParseInfo.mk_adj_mat(EPI, engineJMatLink,
nodesName, intrac);
                        BNDAG=(double[][])temp1.get("intra");
                        nodesName=(String[])temp1.get("names");
                        engineJMatLink.engEvalString(EPI,"nodesNum=length(names)");

                        engineJMatLink.engEvalString(EPI,"figure");
                        engineJMatLink.engEvalString(EPI,"draw_graph(intra,names)");//draw
the BN network graph.

                        for (int i=0;i<nodesName.length;i++){
                                nodesNamePosition.put(nodesName[i], new
Integer(i+1).toString());
                                nodesPositionName.put(new Integer(i+1).toString(),
nodesName[i]);
                        }

                        nodesSize=new double[nodesName.length];
                        for(int i=0;i<nodesName.length;i++){

        nodesSize[i]=Integer.parseInt((String)nodesSizeMap.get(nodesName[i]));
                        }

                        //get neighborsInfoID
                        Set neighborsName=(Set)neighborsInfo.keySet();
                        Iterator itNeighbor=neighborsName.iterator();
                        while(itNeighbor.hasNext()){
                                String neighborI=(String)itNeighbor.next();
                                String[] sharedVars=(String[])neighborsInfo.get(neighborI);
                                double[] sharedVarsID=new double[sharedVars.length];
                                for(int i=0;i<sharedVarsID.length;i++){

        sharedVarsID[i]=Integer.parseInt((String)nodesNamePosition.get(sharedVars[i]));
                                }
                                neighborsInfoID.put(neighborI, sharedVarsID);
                                linkagesEvidence.put(neighborI, new HashSet());

                        }

                        /*now nodesName is in the order from parents to children
                         * the conditional table should be ordered according to the ordered
nodes name
                         * That is important, because FullBNT in Matlab just accept this
way
                         */

        orderedConProbDis=CallMatlabFunsAndParseInfo.OrderConProbDis(nodesNamePosition,nod
esSizeMap,conProbDis);

            //BNMod: moralized graph is stored in the Matlab workspace.
                        Map temp2=CallMatlabFunsAndParseInfo.moralize(EPI, engineJMatLink,
BNDAG);
                        BNMod=(double[][])temp2.get("BNMod");
                        BNTri=BNMod;

                        //initiate eliminate order and draw moralization graph after local
moralization.
                        eliminateOrder=new double[nodesName.length];
                        for(int i=0;i<nodesName.length;i++){
                                eliminateOrder[i]=i+1;
                        }


                        Set neighborBNsAndLJFBUName=new HashSet();
                        neighborBNsAndLJFBUName.addAll(neighborsInfo.keySet());
                        neighborBNsAndLJFBUName.add(ConstantCollection.LJFBUSysAgentName);
                        SenderNameMatchExpression neighborBNsAndLJFBUNameMatch=new
SenderNameMatchExpression(neighborBNsAndLJFBUName);
```

```
                        MessageTemplate mtneighborBNsAndLJFBUNameMatch=new
MessageTemplate(neighborBNsAndLJFBUNameMatch);
                        MessageTemplate
mtInformMatch=MessageTemplate.MatchPerformative(ACLMessage.INFORM);
                        MessageTemplate
mtPropogateMatch=MessageTemplate.MatchPerformative(ACLMessage.PROPAGATE);

        mtForCoTriangulation=MessageTemplate.and(mtneighborBNsAndLJFBUNameMatch,
mtInformMatch);

        mtForCoBeliefUpdate=MessageTemplate.and(mtneighborBNsAndLJFBUNameMatch,
mtPropogateMatch);

                        MessageTemplate tempMT1=MessageTemplate.MatchSender(LJFBUSysAgent);
                        MessageTemplate
tempMT2=MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);
                        mtForTriagulationOver=MessageTemplate.and(tempMT1,tempMT2 );

                        Set componentNames=componentSt.keySet();
                        SenderNameMatchExpression componentNameMatch=new
SenderNameMatchExpression(componentNames);
                        MessageTemplate mtComponentNameMatch=new
MessageTemplate(componentNameMatch);
                        mtForComponent=MessageTemplate.and(mtComponentNameMatch,
mtInformMatch);

                        //get message template for accepting sensor command
                        Set sensorNames=observationOt.keySet();
                        SenderNameMatchExpression sensorNameMatch=new
SenderNameMatchExpression(sensorNames);
                        MessageTemplate mtSensorNameMatch=new
MessageTemplate(sensorNameMatch);
                        mtForSensor=MessageTemplate.and(mtSensorNameMatch, mtInformMatch);

                        addBehaviour(new AcceptMessagesForCoTriangulation(this));//Organize
sub Bayesian networks into certain structures
                        addBehaviour(new AcceptCoTriangulationOverMessage(this));//Indicate
organized structure is ready.
                        addBehaviour(new AcceptEvidenceFromComponent(this));//Accept
previous command information from components
                        addBehaviour(new AcceptEvidenceFromSensors(this));//Accept
information from sensors
                        addBehaviour(new AcceptMessagesForCoBeliefUpdate(this));//Perform
coordinate belief updating and send results to components
                }
        }


        //*************************************************************************
******
        class AcceptMessagesForCoTriangulation extends CyclicBehaviour {
                AcceptMessagesForCoTriangulation(Agent a){
                        super(a);
                }
                public void action() {
                        ACLMessage msg=receive(mtForCoTriangulation);

                        if(msg!=null){
                                String sendersName;
                                sendersName=msg.getSender().getLocalName();
                                if(sendersName.equalsIgnoreCase(LJFBUSysAgentName)){
                                }
                                TransferedMessagesForCoTriangulation tranMes=new
TransferedMessagesForCoTriangulation();
                                try {
                                        tranMes=(TransferedMessagesForCoTriangulation)
msg.getContentObject() ;
                                        String actionType=tranMes.getActionType();
                                        Set  partialNodesEdges=new HashSet();
```

```
        if(sendersName.equalsIgnoreCase(LJFBUSysAgentName)&&((tranMes.getActionType()).equ
alsIgnoreCase("DepthFirstEliminate"))){
                                                getMessageFromASystem=true;

                                        }

                                        if((neighborsInfo.get(sendersName)!=null)){

        partialNodesEdges=tranMes.getPartialNodesEdges();
                                        }

                                        //add partialNodesEdges into the Triagulation Matrix
                                        if(partialNodesEdges.size()==0){
                                        }else{

        BNTri=CallMatlabFunsAndParseInfo.integrateParNodesInfo(nodesName, BNTri,
partialNodesEdges);

                                                engineJMatLink.engPutArray(EPI,"BNTri",
BNTri);
                                        }


        if((tranMes.getActionType()).equalsIgnoreCase("DepthFirstEliminate")){
                                                ActionForDepthFirstEliminate(sendersName);
                                        }

        if((tranMes.getActionType()).equalsIgnoreCase("FillInsFeedBack")){
                                                countReceivedMesOfFillInsFeedBack++;
                                                Set neighbors=neighborsInfo.keySet();
                                                int neighborsNum=neighbors.size();

        if(countReceivedMesOfFillInsFeedBack==(neighborsNum-1)&&!getMessageFromASystem){
                                                        countReceivedMesOfFillInsFeedBack=0;
                                                        ActionForFillInsFeedBack();
                                                }

        if(getMessageFromASystem&&(countReceivedMesOfFillInsFeedBack==neighborsNum)){
                                                        getMessageFromASystem=false;
                                                        recordDFESender=null;
                                                        countReceivedMesOfFillInsFeedBack=0;
                                                        ActionForDistributeDLink(BNName);


                                                        sendMessageAboutFillInExistence();
                                                        existenceOfFillIns="0";

                                                }
                                        }

        if((tranMes.getActionType()).equalsIgnoreCase("DistributeDLink")){
                                                ActionForDistributeDLink(sendersName);
                                                getMessageFromASystem=false;
                                                sendMessageAboutFillInExistence();
                                                existenceOfFillIns="0";

                                        }


                                }catch(Exception ie0){
                                        ie0.printStackTrace();
                                }
                        }else {
                                block();
                        }
                }
        }

        class AcceptCoTriangulationOverMessage extends CyclicBehaviour{
                AcceptCoTriangulationOverMessage(Agent a){
```

```
                        super(a);
                }

                public void action() {
                        ACLMessage msg=receive(mtForTriagulationOver);
                        if(msg!=null){
                                TransferedMessagesForCoTriangulation tranMes=new
TransferedMessagesForCoTriangulation();
                                try {
                                        tranMes=(TransferedMessagesForCoTriangulation)
msg.getContentObject() ;

        if(tranMes.getActionType().equalsIgnoreCase("TriangulationOver")){
                                        System.out.println("In Bayesian network, get
a message about Triangulation Over");
                                        prepareForBeliefUpdating(EPI);//that is
enter prior probability, establish an inference engine before any evidence shows up.
                                        getMessageFromASystem=false;
                                }
                                }catch(Exception ie0){
                                        ie0.printStackTrace();
                                }
                        }else {
                                block();
                        }
                }

        }

        //sensor will provide two information: previous command(hard
evidence),flowState(soft evidence)
        //I need to think about that very carefully.
        class AcceptEvidenceFromSensors extends CyclicBehaviour{
                AcceptEvidenceFromSensors(Agent a) {
                        super(a);
                }
                public void action() {
                        ACLMessage msg=receive(mtForSensor);
                        if(msg!=null){
                                String sendersName;
                                sendersName=msg.getSender().getLocalName();
                                double[] stateDist;
                                try {
                                        stateDist=(double[]) msg.getContentObject();
                                        String
nodeName=(String)observationOt.get(sendersName);
                                        softEvidence.put(nodeName, stateDist);
                                        //System.out.println("++++++++++++: "+sendersName+"
"+nodeName);
                                        //System.out.println("%%%%%%%%%%%%%%%:flowrate
distribution: "+nodeName);
                                        //CallMatlabFunsAndParseInfo.printVector(stateDist);
                                }catch(Exception ie0){
                                        ie0.printStackTrace();
                                }
                        }else {
                                block();
                        }
                }
        }

        //*************************************************************************
*************
        class AcceptEvidenceFromComponent extends CyclicBehaviour{
                AcceptEvidenceFromComponent(Agent a) {
                        super(a);
                }
                //&&&
```

```
                        if(msg!=null){
                                String[] msgArray=(msg.getContent()).split(";");
                                //System.out.println("*********************:  "+msg);
                                String sendersName;
                                sendersName=msg.getSender().getLocalName();

                                        String
nodeNamePreviousCommand=(String)componentCpt1.get(sendersName);
                                        hardEvidence.put(nodeNamePreviousCommand,
msgArray[0]);
                                        //System.out.println("@@@@@@@@@@:"+ msgArray[0]);
                                        if(msgArray[1].equalsIgnoreCase("1")){
                                                double[] openDegreeDist=new double[2];
                                                openDegreeDist[0]=0.9999;
                                                openDegreeDist[1]=0.0001;
                                                String
nodeNameOpenDegree=(String)componentODt.get(sendersName);
                                                if(msgArray[2].equalsIgnoreCase("2"))
                                                {
                                                        openDegreeDist[0]=0.0001;
                                                        openDegreeDist[1]=0.9999;
                                                }
                                                softEvidence.put(nodeNameOpenDegree,
openDegreeDist);
                                                //System.out.println("++++++++++++:
"+sendersName+"  "+nodeNameOpenDegree);
                                                //hardEvidence.put(nodeNameOpenDegree,
msgArray[2]);
                                                //System.out.println("@@@@@@@@@@:"+
msgArray[1]);
                                                //System.out.println("@@@@@@@@@@:"+
msgArray[2]);
                                        }
                        }else {
                                block();
                        }
                }
        }

        //*************************************************************************************
*************
        class UpdateBeliefByLocalEvidence extends TickerBehaviour{

                UpdateBeliefByLocalEvidence(Agent a,long period){
                        super(a, period);
                }
                protected void onTick() {
                        enterLocalEvidences(EPI);
                }

        }

        //*************************************************************************************
*************************
        class AcceptMessagesForCoBeliefUpdate extends CyclicBehaviour {
                AcceptMessagesForCoBeliefUpdate (Agent a){
                        super(a);
                }
                public void action() {
                        ACLMessage msg=receive(mtForCoBeliefUpdate);

                        if(msg!=null){
                                String sendersName;
                                sendersName=msg.getSender().getLocalName();
                                TransferedMessagesForCoBeliefUpdate tranMes=new
TransferedMessagesForCoBeliefUpdate();
                                try {
                                        tranMes=(TransferedMessagesForCoBeliefUpdate)
msg.getContentObject() ;
                                        String actionType=tranMes.getActionType();
```

```
        if(sendersName.equalsIgnoreCase(LJFBUSysAgentName)&&((tranMes.getActionType()).equ
alsIgnoreCase("CollectBelief"))){
                                                getMessageFromASystem=true;
                                        }


        if((tranMes.getActionType()).equalsIgnoreCase("CollectBelief")){
                                                System.out.println(BNName+"  get
CollectBelief "+sendersName);
                                                enterLocalEvidences(EPI);
                                                //Not very sure about that yet.
                                                //In one cycle, the evidence can only be
used once.
                                                //Next cycle, the evidence will be updated.
                                                //hardEvidence.clear();
                                                //softEvidence.clear();
                                                ActionForCollectBelief(EPI,sendersName);

                                        }

        if((tranMes.getActionType()).equalsIgnoreCase("AbsorbThroughLinkage")){
                                                System.out.println(BNName+"  get
AbsorbThroughLinkage "+sendersName);

        ActionForAbsorbThroughLinkage(sendersName,tranMes);
                                        }

        if((tranMes.getActionType()).equalsIgnoreCase("UpdateBelief")){
                                                System.out.println(BNName+"  get
UpdateBelief "+sendersName);
                                                enterLinkageEvidence(EPI,sendersName);
                                                countReceivedUpdateBeliefMessages++;
                                                Set neighbors=neighborsInfo.keySet();
                                                int neighborsNum=neighbors.size();

        if((countReceivedUpdateBeliefMessages==(neighborsInfo.keySet().size()-
1))&&(!getMessageFromASystem))
                                                {
                                                        System.out.println("I am a dummy:
"+BNName+": "+getMessageFromASystem+":  "+recordCollectBeliefSender);
                                                        countReceivedUpdateBeliefMessages=0;

        SendAbsorbThroughLinkageMessages(EPI,recordCollectBeliefSender);

        SendCollectOrDistributeOrUpdateBeliefMessage(recordCollectBeliefSender,"UpdateBeli
ef");
                                                }else
if(countReceivedUpdateBeliefMessages==(neighborsInfo.keySet().size())&&getMessageFromASys
tem){
                                                        System.out.println("I am NOT a dummy:
"+BNName+": "+getMessageFromASystem+":  "+recordCollectBeliefSender);

        ActionForDistributeBelief(EPI,LJFBUSysAgentName);//Actually, here, I assume
LJFBUSysAgent send a distribute belief message.
                                                        getStAndSendMesToComponent(EPI);
                                                        reInitalizeBNNetwork(EPI);
                                                }

                                        }

        if((tranMes.getActionType()).equalsIgnoreCase("DistributeBelief")){
                                                System.out.println(BNName+"  get
DistributeBelief "+sendersName);
                                                ActionForDistributeBelief(EPI,sendersName);
                                                getStAndSendMesToComponent(EPI);
                                                reInitalizeBNNetwork(EPI);
                                        }
                                }catch(Exception ie0){
                                        ie0.printStackTrace();
                                }
```

```
                        }else {
                                block();
                        }
                }
        }

        //*************************************************************************************
*******************
        void ActionForDepthFirstEliminate(String sendersName){
                Set tempNeighbors0=neighborsInfo.keySet();
                Set tempNeighbors1=new HashSet();
                tempNeighbors1.addAll(tempNeighbors0);

                TransferedMessagesForCoTriangulation tranMesRespond=new
TransferedMessagesForCoTriangulation();
                tranMesRespond.setActionType("DepthFirstEliminate");
                //when a BN agent get a DepthFirstEliminate message, if it just has one
neighbor which is the caller,send a FillInsFeedBack messages
                //otherwise, send DepthFirstEliminate message to its neighbors except the
caller
                if(tempNeighbors1.size()==1){

                }else{
                        tempNeighbors1.remove(sendersName);
                }

                Iterator itNeighbors=tempNeighbors1.iterator();
                while(itNeighbors.hasNext()){
                        String tempNeighborName=(String)itNeighbors.next();


        if(tempNeighborName.equalsIgnoreCase(sendersName)&&tempNeighbors1.size()==1){
                                tranMesRespond.setActionType("FillInsFeedBack");

                        }
                        //formulate stage1 and stage2
                        String[] sharedVars=(String[])neighborsInfo.get(tempNeighborName);

                        Map stages=new HashMap();
                        double[] stage2=new double[sharedVars.length];
                        double[] stage1=new double[nodesName.length-stage2.length];

                        for(int i=0;i<stage2.length;i++){

        stage2[i]=Integer.parseInt((String)nodesNamePosition.get(sharedVars[i]));
                        }

                        int countNotSharedVar=0;
                        for(int i=1;i<=nodesName.length;i++){
                                boolean flag=true;
                                for(int j=0;j<stage2.length;j++){
                                        if(i==stage2[j]){
                                                flag=false;
                                                break;
                                        }
                                }
                                if(flag){
                                        stage1[countNotSharedVar]=i;
                                        countNotSharedVar++;
                                }
                        }
                        stages.put("stage1", stage1);
                        stages.put("stage2", stage2);
                        //formulation of stage1 and stage2 is finished

                        Map
cliques_fillIns_tri=CallMatlabFunsAndParseInfo.graph_to_jtree(EPI, engineJMatLink, BNTri,
nodesSize, stages);
                        cliques=(Map)cliques_fillIns_tri.get("cliques");
                        BNTri=(double[][])cliques_fillIns_tri.get("BNTri");
```

```
	existenceOfFillIns=(String)cliques_fillIns_tri.get("existenceOfFillIns");

			Set
sharedVarEdges=CallMatlabFunsAndParseInfo.extractParNodesInfo(nodesName, BNTri,
sharedVars);

			tranMesRespond.setPartialNodesEdges(sharedVarEdges);
			ACLMessage msgToNeighbor=new ACLMessage(ACLMessage.INFORM);
			try {
				msgToNeighbor.setContentObject(tranMesRespond);
				AID receiver=new AID(tempNeighborName,AID.ISLOCALNAME);
				msgToNeighbor.addReceiver(receiver);
				send(msgToNeighbor);
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}


	void ActionForFillInsFeedBack(){
		TransferedMessagesForCoTriangulation tranMesRespond=new
TransferedMessagesForCoTriangulation();
		tranMesRespond.setActionType("FillInsFeedBack");
		//remove  the neighbors which this BN agent has already sent a
DepthFirstEliminate Messages

		//formulate stage1 and stage2
		String[] sharedVars=(String[])neighborsInfo.get(recordDFESender);

		Map stages=new HashMap();
		double[] stage2=new double[sharedVars.length];
		double[] stage1=new double[nodesName.length-stage2.length];

		for(int i=0;i<stage2.length;i++){

	stage2[i]=Integer.parseInt((String)nodesNamePosition.get(sharedVars[i]));
		}

		int countNotSharedVar=0;
		for(int i=1;i<=nodesName.length;i++){
			boolean flag=true;
			for(int j=0;j<stage2.length;j++){
				if(i==stage2[j]){
					flag=false;
					break;
				}
			}
			if(flag){
				stage1[countNotSharedVar]=i;
				countNotSharedVar++;
			}
		}
		stages.put("stage1", stage1);
		stages.put("stage2", stage2);
		//formulation of stage1 and stage2 is finished

		Map cliques_fillIns_tri=CallMatlabFunsAndParseInfo.graph_to_jtree(EPI,
engineJMatLink, BNTri, nodesSize, stages);
		cliques=(Map)cliques_fillIns_tri.get("cliques");
		BNTri=(double[][])cliques_fillIns_tri.get("BNTri");
		existenceOfFillIns=(String)cliques_fillIns_tri.get("existenceOfFillIns");

		Set
sharedVarEdges=CallMatlabFunsAndParseInfo.extractParNodesInfo(nodesName, BNTri,
sharedVars);

		tranMesRespond.setPartialNodesEdges(sharedVarEdges);
		ACLMessage msgToNeighbor=new ACLMessage(ACLMessage.INFORM);
		try {
```

```
                        msgToNeighbor.setContentObject(tranMesRespond);
                        AID receiver=new AID(recordDFESender,AID.ISLOCALNAME);
                        msgToNeighbor.addReceiver(receiver);
                        send(msgToNeighbor);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }


        void ActionForDistributeDLink(String sendersName){
                Set tempNeighbors0=neighborsInfo.keySet();
                Set tempNeighbors1=new HashSet();
                tempNeighbors1.addAll(tempNeighbors0);

                Iterator itNeighbors=tempNeighbors1.iterator();
                TransferedMessagesForCoTriangulation tranMesRespond=new
TransferedMessagesForCoTriangulation();
                tranMesRespond.setActionType("DistributeDLink");
                while(itNeighbors.hasNext()){
                        String tempNeighborName=(String)itNeighbors.next();

                        if(!tempNeighborName.equalsIgnoreCase(sendersName)){
                                //formulate stage1 and stage2
                                String[]
sharedVars=(String[])neighborsInfo.get(tempNeighborName);
                                Map stages=new HashMap();
                                double[] stage2=new double[sharedVars.length];
                                double[] stage1=new double[nodesName.length-stage2.length];

                                for(int i=0;i<stage2.length;i++){

        stage2[i]=Integer.parseInt((String)nodesNamePosition.get(sharedVars[i]));
                                }

                                int countNotSharedVar=0;
                                for(int i=1;i<=nodesName.length;i++){
                                        boolean flag=true;
                                        for(int j=0;j<stage2.length;j++){
                                                if(i==stage2[j]){
                                                        flag=false;
                                                        break;
                                                }
                                        }
                                        if(flag){
                                                stage1[countNotSharedVar]=i;
                                                countNotSharedVar++;
                                        }
                                }
                                stages.put("stage1", stage1);
                                stages.put("stage2", stage2);
                                //formulation of stage1 and stage2 is finished

                                Map
cliques_fillIns_tri=CallMatlabFunsAndParseInfo.graph_to_jtree(EPI, engineJMatLink, BNTri,
nodesSize, stages);
                                cliques=(Map)cliques_fillIns_tri.get("cliques");
                                BNTri=(double[][])cliques_fillIns_tri.get("BNTri");

        existenceOfFillIns=(String)cliques_fillIns_tri.get("existenceOfFillIns");

                                Set
sharedVarEdges=CallMatlabFunsAndParseInfo.extractParNodesInfo(nodesName, BNTri,
sharedVars);

                                tranMesRespond.setPartialNodesEdges(sharedVarEdges);
                                ACLMessage msgToNeighbor=new ACLMessage(ACLMessage.INFORM);
                                try {
                                        msgToNeighbor.setContentObject(tranMesRespond);
                                        AID receiver=new
AID(tempNeighborName,AID.ISLOCALNAME);
```

```
                                        msgToNeighbor.addReceiver(receiver);
                                        send(msgToNeighbor);
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }

                        }
                }
        }


        void sendMessageAboutFillInExistence()
        {
                ACLMessage messageAboutFillInExistence=new ACLMessage(ACLMessage.INFORM);
                messageAboutFillInExistence.setContent(existenceOfFillIns);
                messageAboutFillInExistence.addReceiver(LJFBUSysAgent);
                send(messageAboutFillInExistence);
        }



        void prepareForBeliefUpdating(long EPI){
                double MatLabflag=0;
                /*
                 * after that step, the coordinate triangulation for this BN network is
done.
                 * it is time to establish the junction forest
                 * the triangulated graph information will be stored in Matlab workspace,
                 * so I do not need to establish an inference engine every time
                 * but the evidence will be changed all the time
                 * Now we have: intra(Ordered DAG), names(ordered nodes'
names),BNTri(coordinated triangulated graph), ns(nodes size)
                 */
         engineJMatLink.engEvalString(EPI,"MatLabflag=0");
                engineJMatLink.engEvalString(EPI,"bnet=mk_bnet(intra,ns,'names',names)");

                for(int i=0;i<nodesName.length;i++){
                        Map tempTableMap=(Map)orderedConProbDis.get(nodesName[i]);
                        String[]
tempNodesNameInPrepare=(String[])tempTableMap.get("nodesName");
                        int switchNum=tempNodesNameInPrepare.length;
                        switch (switchNum) {
                        case 1: double[] table1=(double[])tempTableMap.get("conProbDis");
                        double[] tempTable1=table1;
                        engineJMatLink.engPutArray(EPI,"conTable", tempTable1);
                        int temp1=i+1;

        engineJMatLink.engEvalString(EPI,"bnet.CPD{"+temp1+"}=tabular_CPD(bnet,"+temp1+",c
onTable)" );
                        break;

                        case 2: double[][]
table2=(double[][])tempTableMap.get("conProbDis");
                        double[]
tempTable2=CallMatlabFunsAndParseInfo.multiDimToVector(table2);
                        engineJMatLink.engPutArray(EPI,"conTable", tempTable2);
                        int temp2=i+1;

        engineJMatLink.engEvalString(EPI,"bnet.CPD{"+temp2+"}=tabular_CPD(bnet,"+temp2+",c
onTable)" );
                        break;

                        case 3: double[][][]
table3=(double[][][])tempTableMap.get("conProbDis");
                        double[]
tempTable3=CallMatlabFunsAndParseInfo.multiDimToVector(table3);
                        engineJMatLink.engPutArray(EPI,"conTable", tempTable3);
                        int temp3=i+1;
```

```
        engineJMatLink.engEvalString(EPI,"bnet.CPD{"+temp3+"}=tabular_CPD(bnet,"+temp3+",c
onTable)" );
                        break;

                        case 4: double[][][][]
table4=(double[][][][])tempTableMap.get("conProbDis");
                        double[]
tempTable4=CallMatlabFunsAndParseInfo.multiDimToVector(table4);
                        engineJMatLink.engPutArray(EPI,"conTable", tempTable4);
                        int temp4=i+1;

        engineJMatLink.engEvalString(EPI,"bnet.CPD{"+temp4+"}=tabular_CPD(bnet,"+temp4+",c
onTable)" );
                        break;

                        case 5:
                        double[][][][][]
table5=(double[][][][][])tempTableMap.get("conProbDis");
                        double[]
tempTable5=CallMatlabFunsAndParseInfo.multiDimToVector(table5);
                        engineJMatLink.engPutArray(EPI,"conTable", tempTable5);
                        int temp5=i+1;

        engineJMatLink.engEvalString(EPI,"bnet.CPD{"+temp5+"}=tabular_CPD(bnet,"+temp5+",c
onTable)" );
                        break;

                        default: System.out.println("the conditional table dimension can
not be zero or >5");System.exit(0);
                        }
                }

        engineJMatLink.engEvalString(EPI,"engineOld=jtree_inf_engine(bnet,'triangulatedG',
BNTri)");
                engineJMatLink.engEvalString(EPI,"cliques=cliques_from_engine(engineOld)");
                engineJMatLink.engEvalString(EPI,"len=length(cliques)");

                engineJMatLink.engEvalString(EPI,"MatLabflag=1");

                MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                while(MatLabflag==0){
                        MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                }

                double[][] lenTemp;
                lenTemp=engineJMatLink.engGetArray(EPI, "len");
                double len=lenTemp[0][0];

                Map cliquesFinal=new HashMap();
                //extract cliques from Matlab to a Map
                for(int i=1;i<=len;i++){
                        engineJMatLink.engEvalString(EPI,"MatLabflag=0");
                        String temp="cliquesI=cliques{"+i+"}";
                        engineJMatLink.engEvalString(EPI,temp);

                        engineJMatLink.engEvalString(EPI,"MatLabflag=1");

                        MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                        while(MatLabflag==0){
                                MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                        }

                        double[][] tempCC=engineJMatLink.engGetArray(EPI, "cliquesI");
                        double[] tempC=tempCC[0];

                        cliquesFinal.put(new Integer(i).toString(), tempC);
                }
                cliques=cliquesFinal;
                engineJMatLink.engEvalString(EPI,"evidence=cell(1,length(names))");
```

```
        engineJMatLink.engEvalString(EPI,"[engineOld,loglik]=enter_evidence(engineOld,evid
ence)");
                engineJMatLink.engEvalString(EPI,"engineNew=engineOld");
        }


        void enterLocalEvidences(long EPI){
                int nodesNum=nodesName.length;

                //Put hardEvidence to Matlab workspace.
                engineJMatLink.engEvalString(EPI,"hardEvidence=cell(1,nodesNum)");
                for(int i=0;i<nodesNum;i++){
                        double hardEvidenceI;
                        String hardEvidenceITemp=(String)hardEvidence.get(nodesName[i]);
                        if(hardEvidenceITemp!=null){
                                //System.out.println("^^^^^^^^^^^^^^^^^^^:
"+hardEvidenceITemp);
                                hardEvidenceI=Double.parseDouble(hardEvidenceITemp);
                                int nodesI=i+1;
                                engineJMatLink.engPutArray(EPI,"nodesI", nodesI);
                                engineJMatLink.engPutArray(EPI,"hardEvidenceI",
hardEvidenceI);

        engineJMatLink.engEvalString(EPI,"hardEvidence{nodesI}=hardEvidenceI");
                                //System.out.println("^^^^^^^hardEvidence:
"+nodesName[i]+"    "+hardEvidenceI);
                        }
                }
                //hardEvidence.clear();
                //Put softEvidence to Matlab workspace.//softEvidence is a
Map(key=nodename,value=double[]prob distribution)
                engineJMatLink.engEvalString(EPI,"softEvidence=cell(1,nodesNum)");
                for(int i=0;i<nodesNum;i++){
                        double[] softEvidenceI;
                        if((softEvidenceI=(double[])softEvidence.get(nodesName[i]))!=null){
                                int nodesI=i+1;
                                engineJMatLink.engPutArray(EPI,"nodesI", nodesI);
                                engineJMatLink.engPutArray(EPI,"softEvidenceI",
softEvidenceI);

        engineJMatLink.engEvalString(EPI,"softEvidence{nodesI}=softEvidenceI");
                                //System.out.println("^^^^^^^softEvidence:
"+nodesName[i]);
                                //CallMatlabFunsAndParseInfo.printVector(softEvidenceI);
                        }
                }

        engineJMatLink.engEvalString(EPI,"[engineNew,loglikNew]=enter_evidence(engineNew,h
ardEvidence,'soft',softEvidence)");
        }


        void enterLinkageEvidence(long EPI, String neighborName){
                engineJMatLink.engEvalString(EPI,
"[clpot,seppot]=get_clpot_seppot(engineNew)");
                double[] sharedVars=(double[])neighborsInfoID.get(neighborName);

        engineJMatLink.engEvalString(EPI,"[cliques,separators]=get_cliques_separators(engi
neNew)");
                engineJMatLink.engPutArray(EPI,"sharedVars", sharedVars);

        engineJMatLink.engEvalString(EPI,"[hostCliqueID,linkages,peerSepset,linkageSeparat
or]=get_linkagetree(cliques,separators,sharedVars)");
                Set tmfcbu=(Set)linkagesEvidence.get(neighborName);
                Iterator it0=tmfcbu.iterator();

                while(it0.hasNext()){
                        TransferedMessagesForCoBeliefUpdate
tmfcbuI=(TransferedMessagesForCoBeliefUpdate)it0.next();
                        String[] linkageINodesName=tmfcbuI.getLinkageDomain();
```

```
                        double[] linkageIDistVec=tmfcbuI.getLinkageDistribution();
                        engineJMatLink.engPutArray(EPI,"linkageIDistVec", linkageIDistVec);

                        int linkageINodesNum=linkageINodesName.length;
                        double[] linkageIDomain=new double[linkageINodesNum];
                        double[] linkageISizes=new double[linkageINodesNum];
                        for(int i=0;i<linkageINodesNum;i++){
                                int
tempIDI=Integer.parseInt((String)nodesNamePosition.get(linkageINodesName[i]));
                                linkageIDomain[i]=tempIDI;
                                linkageISizes[i]=nodesSize[tempIDI-1];
                        }
                        engineJMatLink.engPutArray(EPI,"linkageIDomain", linkageIDomain);
                        engineJMatLink.engPutArray(EPI,"linkageISizes", linkageISizes);

        engineJMatLink.engEvalString(EPI,"linkageIPotNew=get_linkageI_pot(linkageIDomain,l
inkageISizes,linkageIDistVec)");

        engineJMatLink.engEvalString(EPI,"clpot=update_clpot(clpot,linkageIPotNew,hostCliq
ueID,linkages,peerSepset,engineNew)");
                }
                engineJMatLink.engEvalString(EPI,"[engineNew, loglikNew] =
enter_cliques_evidence(engineNew,clpot,seppot)");
                tmfcbu.clear();
                linkagesEvidence.put(neighborName, tmfcbu);
        }


        //just put the AbsorbThroughLinkage message into the corresponded neighbor's
linkages information set.
        void ActionForAbsorbThroughLinkage(String
neighborName,TransferedMessagesForCoBeliefUpdate mesForCBU){
                Set temp1=(Set)linkagesEvidence.get(neighborName);
                temp1.add(mesForCBU);
                linkagesEvidence.put(neighborName, temp1);
        }



        void ActionForCollectBelief(long EPI,String sendersName){
                recordCollectBeliefSender=sendersName;

                Set tempNeighbors0=neighborsInfo.keySet();
                Set tempNeighbors1=new HashSet();
                tempNeighbors1.addAll(tempNeighbors0);

                if(sendersName.equals(LJFBUSysAgentName)){
                        Iterator itNeighbors0=tempNeighbors0.iterator();
                        while(itNeighbors0.hasNext()){
                                String tempNeighborName0=(String)itNeighbors0.next();

        SendCollectOrDistributeOrUpdateBeliefMessage(tempNeighborName0,"CollectBelief");
                        }
                        return;
                }

                if(tempNeighbors1.size()==1){

                }else{
                        tempNeighbors1.remove(sendersName);
                }

                Iterator itNeighbors=tempNeighbors1.iterator();
                while(itNeighbors.hasNext()){
                        String tempNeighborName=(String)itNeighbors.next();

                        //For CollectBelief, if T is called to operate CollectBelief,
                        //if T has no neighbor except caller, it performs UnifyBelief and
return (send linkage distribution to the caller)
                        //Otherwise, for each adjacent JTBU Y except caller, call
CollectBelief in Y.
```

```
        if(tempNeighborName.equalsIgnoreCase(sendersName)&&tempNeighbors1.size()==1){
                                SendAbsorbThroughLinkageMessages(EPI,sendersName);

        SendCollectOrDistributeOrUpdateBeliefMessage(sendersName,"UpdateBelief");
                        }else{

        SendCollectOrDistributeOrUpdateBeliefMessage(tempNeighborName,"CollectBelief");
                        }
                }
        }

        void SendAbsorbThroughLinkageMessages(long EPI,String receiverName)
        {
                double MatLabflag=0;
                engineJMatLink.engEvalString(EPI,"MatLabFlag=0");
                engineJMatLink.engEvalString(EPI,
"[clpot,seppot]=get_clpot_seppot(engineNew)");
                double[] sharedVars=(double[])neighborsInfoID.get(receiverName);

        engineJMatLink.engEvalString(EPI,"[cliques,separators]=get_cliques_separators(engi
neNew)");
                engineJMatLink.engPutArray(EPI,"sharedVars", sharedVars);

        engineJMatLink.engEvalString(EPI,"[hostCliqueID,linkages,peerSepset,linkageSeparat
or]=get_linkagetree(cliques,separators,sharedVars)");
                engineJMatLink.engEvalString(EPI,"numOfLinkages=length(linkages)");

                engineJMatLink.engEvalString(EPI,"MatLabflag=1");

                MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                while(MatLabflag==0){
                        MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabFlag");
                }

                double numOfLinkages=engineJMatLink.engGetScalar(EPI,"numOfLinkages");

                int num=(int)numOfLinkages;
                for(int i=1;i<=num;i++){
                        double linkageIIndex=i;
                        engineJMatLink.engEvalString(EPI,"MatLabflag=0");
                        engineJMatLink.engPutArray(EPI,"linkageIIndex", linkageIIndex);

        engineJMatLink.engEvalString(EPI,"[linkageIDomain,linkageIDistVec]=get_linkageI_Di
stVec(linkageIIndex,linkages,peerSepset,engineNew)");

                        engineJMatLink.engEvalString(EPI,"MatLabflag=1");

                        MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                        while(MatLabflag==0){
                                MatLabflag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                        }

                        double[] linkageIDomain=engineJMatLink.engGetArray(EPI,
"linkageIDomain")[0];
                        double[]
linkageIDistVec=engineJMatLink.engGetArray(EPI,"linkageIDistVec")[0];

                        int len=linkageIDomain.length;
                        String[] linkageINodesName=new String[len];
                        for(int j=0;j<len;j++){
                                int temp0=(int)linkageIDomain[j];
                                linkageINodesName[j]=(String)nodesPositionName.get(new
Integer(temp0).toString());
                        }

                        TransferedMessagesForCoBeliefUpdate tranMesRespondForLinkage=new
TransferedMessagesForCoBeliefUpdate();
                        tranMesRespondForLinkage.setActionType("AbsorbThroughLinkage");
                        tranMesRespondForLinkage.setLinkageDomain(linkageINodesName);
                        tranMesRespondForLinkage.setLinkageDistribution(linkageIDistVec);
```

```
                        ACLMessage msgToNeighborForLinkage=new
ACLMessage(ACLMessage.PROPAGATE);
                        try {

        msgToNeighborForLinkage.setContentObject(tranMesRespondForLinkage);
                                AID receiver=new AID(receiverName,AID.ISLOCALNAME);
                                msgToNeighborForLinkage.addReceiver(receiver);
                                send(msgToNeighborForLinkage);
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }


        void SendCollectOrDistributeOrUpdateBeliefMessage(String receiverName,String
actionType){
                TransferedMessagesForCoBeliefUpdate tranMesRespondForCollectBelief=new
TransferedMessagesForCoBeliefUpdate();
                tranMesRespondForCollectBelief.setActionType(actionType);
                ACLMessage msgToNeighborForDistributeBelief=new
ACLMessage(ACLMessage.PROPAGATE);
                try {

        msgToNeighborForDistributeBelief.setContentObject(tranMesRespondForCollectBelief);
                        AID receiver=new AID(receiverName,AID.ISLOCALNAME);
                        msgToNeighborForDistributeBelief.addReceiver(receiver);
                        send(msgToNeighborForDistributeBelief);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        void ActionForDistributeBelief(long EPI,String sendersName){
                if(sendersName.equals(LJFBUSysAgentName)){
                        //do not need to unify belief
                }else{
                        enterLinkageEvidence(EPI, sendersName);
                }
                Set tempNeighbors=neighborsInfo.keySet();

                Iterator itNeighbors=tempNeighbors.iterator();
                while(itNeighbors.hasNext()){
                        String tempNeighborName=(String)itNeighbors.next();
                        if(tempNeighborName.equalsIgnoreCase(sendersName)){

                        }else{
                                SendAbsorbThroughLinkageMessages(EPI,tempNeighborName);

        SendCollectOrDistributeOrUpdateBeliefMessage(tempNeighborName,"DistributeBelief");
                        }
                }
        }

        void getStAndSendMesToComponent(long EPI){
                double MatLabFlag=0;
                engineJMatLink.engEvalString(EPI,"MatLabflag=0");
         Set componentNames=componentSt.keySet();
         Iterator itComponentNames=componentNames.iterator();

         //clear previous hard evidence and soft evidence
         hardEvidence.clear();
         softEvidence.clear();


         while(itComponentNames.hasNext()){
                ACLMessage mesToComponent=new ACLMessage(ACLMessage.INFORM);
                String componentNameString=(String)itComponentNames.next();
                String nodeName=(String)componentSt.get(componentNameString);
```

```
                double
componentStIID=Double.parseDouble((String)nodesNamePosition.get(nodeName));
                engineJMatLink.engPutArray(EPI,"componentStIID", componentStIID);

        engineJMatLink.engEvalString(EPI,"componentStIIDMarg=marginal_nodes(engineNew,comp
onentStIID,1)");

        engineJMatLink.engEvalString(EPI,"componentStIDist=(componentStIIDMarg.T)'");


                engineJMatLink.engEvalString(EPI,"MatLabflag=1");

                MatLabFlag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                while(MatLabFlag==0){
                        MatLabFlag=engineJMatLink.engGetScalar(EPI,"MatLabflag");
                }

                double[][]
componentStIDistTemp=engineJMatLink.engGetArray(EPI,"componentStIDist");
                double[] componentStIDist=componentStIDistTemp[0];
                String nodeNamePrevious=nodeName.substring(2);
                nodeNamePrevious="Spt"+nodeNamePrevious;
                softEvidence.put(nodeNamePrevious,componentStIDist);
                try{
                        mesToComponent.setContentObject(componentStIDist);
                        AID componentAgent=new AID(componentNameString,AID.ISLOCALNAME);

                        System.out.println(componentNameString);
                        CallMatlabFunsAndParseInfo.printVector(componentStIDist);

                mesToComponent.addReceiver(componentAgent);
                send(mesToComponent);
                }catch(Exception ie){
                        ie.printStackTrace();
                }
                //System.out.println("++++++++++++: "+componentNameString+"  "+nodeName);
         }


        }

        void reInitalizeBNNetwork(long EPI){
                getMessageFromASystem=false;
                countReceivedMesOfFillInsFeedBack=0;
                countReceivedUpdateBeliefMessages=0;
                recordDFESender=null;
                recordCollectBeliefSender=null;
        }
}
```

```
package bayesianNetworkAgent;

import java.util.*;

import msbn.util.BNInfo;
import msbn.util.ConditionalProbTypes;


public class BayesianNetworkInfoInput {

        public static BNInfo[] getBNNetworks(){
                String[]
BNNetworkAgentName={"resourceBNNetwork","serviceloadBNNetwork1","serviceloadBNNetwork2"};
                int BNNum=BNNetworkAgentName.length;
                BNInfo[] BNNetworkInfo=new BNInfo[BNNum];
```

```
                String[]
nodesName0={"f1","f2","f3","f4","f5","f6","f7","f8","f9","f10","f11","f12","f13","f14","f
17","f18","f21",

        "St1","St2","St3","St4","St5","St6","St7","St8","St9","St10","St11","St12","St13",

        "Spt1","Spt2","Spt3","Spt4","Spt5","Spt6","Spt7","Spt8","Spt9","Spt10","Spt11","Sp
t12","Spt13",

        "Cpt1","Cpt2","Cpt3","Cpt4","Cpt5","Cpt6","Cpt7","Cpt8","Cpt9","Cpt10","Cpt11","Cp
t12","Cpt13",

        "ODt1","ODt2","ODt3","ODt4","ODt5","ODt6","ODt7","ODt8","ODt9","ODt10","ODt11","OD
t12","ODt13"
                };
                String[][]
intrac0={{"f2","f1"},{"f2","f3"},{"f2","f4"},{"f3","f5"},{"f4","f6"},{"f5","f7"},{"f6","f
7"},{"f7","f8"},{"f7","f12"},

        {"f8","f14"},{"f8","f10"},{"f9","f1"},{"f10","f14"},{"f10","f18"},{"f11","f9"},{"f
11","f13"},{"f12","f18"},{"f12","f10"},

        {"f13","f1"},{"f17","f9"},{"f17","f11"},{"f21","f11"},{"f21","f13"},

        {"St1","f1"},{"St2","f2"},{"St3","f3"},{"St4","f4"},{"St5","f5"},{"St6","f6"},{"St
7","f7"},{"St8","f8"},{"St9","f9"},
                                {"St10","f10"},{"St11","f11"},{"St12","f12"},{"St13","f13"},

        {"Spt1","St1"},{"Spt2","St2"},{"Spt3","St3"},{"Spt4","St4"},{"Spt5","St5"},{"Spt6"
,"St6"},{"Spt7","St7"},{"Spt8","St8"},

       {"Spt9","St9"},{"Spt10","St10"},{"Spt11","St11"},{"Spt12","St12"},{"Spt13","St13"},

        {"Cpt1","St1"},{"Cpt2","St2"},{"Cpt3","St3"},{"Cpt4","St4"},{"Cpt5","St5"},{"Cpt6"
,"St6"},{"Cpt7","St7"},{"Cpt8","St8"},

       {"Cpt9","St9"},{"Cpt10","St10"},{"Cpt11","St11"},{"Cpt12","St12"},{"Cpt13","St13"},

        {"St1","ODt1"},{"St2","ODt2"},{"St3","ODt3"},{"St4","ODt4"},{"St5","ODt5"},{"St6",
"ODt6"},{"St7","ODt7"},{"St8","ODt8"},

        {"St9","ODt9"},{"St10","ODt10"},{"St11","ODt11"},{"St12","ODt12"},{"St13","ODt13"}
};
                BNNetworkInfo[0]=new BNInfo(nodesName0,intrac0);

                String[] nodesName1={"f14","f15","f16","f17",
                                "St15","St16",
                                "Spt15","Spt16",
                                "Cpt15","Cpt16",
                                "ODt15","ODt16"
                };
                String[][]
intrac1={{"f14","f15"},{"f14","f16"},{"f15","f17"},{"f16","f17"},
                                {"St15","f15"},{"St16","f16"},
                                {"Spt15","St15"},{"Spt16","St16"},
                                {"Cpt15","St15"},{"Cpt16","St16"},
                                {"St15","ODt15"},{"St16","ODt16"}};
                BNNetworkInfo[1]=new BNInfo(nodesName1,intrac1);


                String[] nodesName2={"f18","f19","f20","f21",
                                "St18","St19",
                                "Spt18","Spt19",
                                "Cpt18","Cpt19",
                                "ODt18","ODt19"
                };
                String[][]
intrac2={{"f18","f19"},{"f18","f20"},{"f19","f21"},{"f20","f21"},
                                {"St18","f19"},{"St19","f20"},
                                {"Spt19","St19"},{"Spt18","St18"},
                                {"Cpt19","St19"},{"Cpt18","St18"},
```

```
                {"St19","ODt19"},{"St18","ODt18"}};
BNNetworkInfo[2]=new BNInfo(nodesName2,intrac2);

//BNNetwork 0, resourceBNNetwork
Map conProbDis0=new HashMap();

Map tableMapf1=new HashMap();
String[] nodesNamesf1={"St1","f9","f13","f2","f1"};
tableMapf1.put("nodesName",nodesNamesf1);
tableMapf1.put("conProbDis", ConditionalProbTypes.conProb14);
conProbDis0.put("f1", tableMapf1);

Map tableMapf2=new HashMap();
String[] nodesNamesf2={"St2","f2"};
tableMapf2.put("nodesName",nodesNamesf2);
tableMapf2.put("conProbDis",ConditionalProbTypes.conProb15);
conProbDis0.put("f2", tableMapf2);

Map tableMapf3=new HashMap();
String[] nodesNamesf3={"St3","f2","f3"};
tableMapf3.put("nodesName",nodesNamesf3);
tableMapf3.put("conProbDis",ConditionalProbTypes.conProb10);
conProbDis0.put("f3", tableMapf3);

Map tableMapf4=new HashMap();
String[] nodesNamesf4={"St4","f2","f4"};
tableMapf4.put("nodesName",nodesNamesf4);
tableMapf4.put("conProbDis",ConditionalProbTypes.conProb10);
conProbDis0.put("f4", tableMapf4);

Map tableMapf5=new HashMap();
String[] nodesNamesf5={"St5","f3","f5"};
tableMapf5.put("nodesName",nodesNamesf5);
tableMapf5.put("conProbDis",ConditionalProbTypes.conProb9);
conProbDis0.put("f5", tableMapf5);

Map tableMapf6=new HashMap();
String[] nodesNamesf6={"St6","f4","f6"};
tableMapf6.put("nodesName",nodesNamesf6);
tableMapf6.put("conProbDis",ConditionalProbTypes.conProb9);
conProbDis0.put("f6", tableMapf6);

Map tableMapf7=new HashMap();
String[] nodesNamesf7={"St7","f5","f6","f7"};
tableMapf7.put("nodesName",nodesNamesf7);
tableMapf7.put("conProbDis",ConditionalProbTypes.conProb4);
conProbDis0.put("f7", tableMapf7);

Map tableMapf8=new HashMap();
String[] nodesNamesf8={"St8","f7","f8"};
tableMapf8.put("nodesName",nodesNamesf8);
tableMapf8.put("conProbDis",ConditionalProbTypes.conProb10);
conProbDis0.put("f8", tableMapf8);

Map tableMapf9=new HashMap();
String[] nodesNamesf9={"St9","f17","f11","f9"};
tableMapf9.put("nodesName",nodesNamesf9);
tableMapf9.put("conProbDis",ConditionalProbTypes.conProb8);
conProbDis0.put("f9", tableMapf9);

Map tableMapf10=new HashMap();
String[] nodesNamesf10={"St10","f8","f12","f10"};
tableMapf10.put("nodesName",nodesNamesf10);
tableMapf10.put("conProbDis",ConditionalProbTypes.conProb5);
conProbDis0.put("f10", tableMapf10);

Map tableMapf11=new HashMap();
String[] nodesNamesf11={"St11","f17","f21","f11"};
tableMapf11.put("nodesName",nodesNamesf11);
tableMapf11.put("conProbDis",ConditionalProbTypes.conProb6);
conProbDis0.put("f11", tableMapf11);
```

```
Map tableMapf12=new HashMap();
String[] nodesNamesf12={"St12","f7","f12"};
tableMapf12.put("nodesName",nodesNamesf12);
tableMapf12.put("conProbDis",ConditionalProbTypes.conProb10);
conProbDis0.put("f12", tableMapf12);

Map tableMapf13=new HashMap();
String[] nodesNamesf13={"St13","f21","f11","f13"};
tableMapf13.put("nodesName",nodesNamesf13);
tableMapf13.put("conProbDis",ConditionalProbTypes.conProb7);
conProbDis0.put("f13", tableMapf13);

Map tableMapf14_0=new HashMap();
String[] nodesNamesf14_0={"f8","f10","f14"};
tableMapf14_0.put("nodesName",nodesNamesf14_0);
tableMapf14_0.put("conProbDis",ConditionalProbTypes.conProb12);
conProbDis0.put("f14", tableMapf14_0);

Map tableMapf17_0=new HashMap();
String[] nodesNamesf17_0={"f17"};
tableMapf17_0.put("nodesName",nodesNamesf17_0);
double[] tempConProb17_0={0.5,0.5};
tableMapf17_0.put("conProbDis",tempConProb17_0.clone());
conProbDis0.put("f17", tableMapf17_0);

Map tableMapf18_0=new HashMap();
String[] nodesNamesf18_0={"f12","f10","f18"};
tableMapf18_0.put("nodesName",nodesNamesf18_0);
tableMapf18_0.put("conProbDis",ConditionalProbTypes.conProb11);
conProbDis0.put("f18", tableMapf18_0);

Map tableMapf21_0=new HashMap();
String[] nodesNamesf21_0={"f21"};
double[] tempConProb21_0={0.5,0.5};
tableMapf21_0.put("nodesName",nodesNamesf21_0);
tableMapf21_0.put("conProbDis",tempConProb21_0.clone());
conProbDis0.put("f21", tableMapf21_0);

for(int i=0;i<nodesName0.length;i++){
        if(nodesName0[i].indexOf("Spt")!=-1){
                Map tableMap=new HashMap();
                String temp=nodesName0[i];
                String[] nodesNameI={temp};
                tableMap.put("nodesName",nodesNameI);
                tableMap.put("conProbDis", ConditionalProbTypes.conProb1);
                conProbDis0.put(nodesName0[i], tableMap);
        }else if(nodesName0[i].indexOf("Cpt")!=-1){
                Map tableMap=new HashMap();
                String temp=nodesName0[i];
                String[] nodesNameI={temp};
                tableMap.put("nodesName",nodesNameI);
                tableMap.put("conProbDis", ConditionalProbTypes.conProb2);
                conProbDis0.put(nodesName0[i], tableMap);
        }else if(nodesName0[i].indexOf("St")!=-1){
                Map tableMap=new HashMap();
                String temp1=nodesName0[i].substring(2);
                String[] nodesNamesSt={"Spt"+temp1,"Cpt"+temp1,"St"+temp1};
                tableMap.put("nodesName",nodesNamesSt);
                tableMap.put("conProbDis", ConditionalProbTypes.conProb3);
                conProbDis0.put(nodesName0[i], tableMap);
        }else if(nodesName0[i].indexOf("ODt")!=-1){
                Map tableMap=new HashMap();
                String temp1=nodesName0[i].substring(3);
                String[] nodesNamesSt={"St"+temp1,"ODt"+temp1};
                tableMap.put("nodesName",nodesNamesSt);
                tableMap.put("conProbDis", ConditionalProbTypes.conProb16);
                conProbDis0.put(nodesName0[i], tableMap);
        }
}
```

```
		Map nodeSize0=new HashMap();
		for(int i=0;i<nodesName0.length;i++){
			if(nodesName0[i].indexOf("f")!=-1){
				nodeSize0.put(nodesName0[i], "2");
			}else if((nodesName0[i].indexOf("St")!=-
1)||(nodesName0[i].indexOf("Spt")!=-1)){
				nodeSize0.put(nodesName0[i], "4");
			}else {
				nodeSize0.put(nodesName0[i], "2");
			}
		}
		nodeSize0.put("f10", "3");
		nodeSize0.put("f11", "3");

		Map neighborsInfo0=new HashMap();
		String[] shared01={"f14","f17"};
		neighborsInfo0.put(BNNetworkAgentName[1], shared01);
		String[] shared02={"f18","f21"};
		neighborsInfo0.put(BNNetworkAgentName[2], shared02);

		Map componentSt0Map=new HashMap();
		Map componentSpt0Map=new HashMap();
		Map componentCpt0Map=new HashMap();
		Map componentODt0Map=new HashMap();

		String[]
componentNames0={"RSADSys_chillWaterResource1_valve1","RSADSys_chillWaterResource1_valve2
","RSADSys_chillWaterResource1_valve3","RSADSys_chillWaterResource1_valve4",

	"RSADSys_chillWaterResource1_pump1","RSADSys_chillWaterResource1_pump2","RSADSys_c
hillWaterResource1_chiller1",

	"RSADSys_crossValvesSystem_coolvalve1","RSADSys_crossValvesSystem_coolvalve2","RSA
DSys_crossValvesSystem_coolvalve3",

	"RSADSys_crossValvesSystem_hotvalve1","RSADSys_crossValvesSystem_hotvalve2","RSADS
ys_crossValvesSystem_hotvalve3"};

		String[] componentSt0={"St4","St3","St1","St7",
				"St6","St5","St2",
				"St8","St10","St12",
				"St9","St11","St13"};
		String[] componentSpt0={"Spt4","Spt3","Spt1","Spt7",
				"Spt6","Spt5","Spt2",
				"Spt8","Spt10","Spt12",
				"Spt9","Spt11","Spt13"};
		String[] componentCpt0={"Cpt4","Cpt3","Cpt1","Cpt7",
				"Cpt6","Cpt5","Cpt2",
				"Cpt8","Cpt10","Cpt12",
				"Cpt9","Cpt11","Cpt13"};
		String[] componentODt0={"ODt4","ODt3","ODt1","ODt7",
				"ODt6","ODt5","ODt2",
				"ODt8","ODt10","ODt12",
				"ODt9","ODt11","ODt13"};

		for(int i=0;i<componentNames0.length;i++){
			componentSt0Map.put(componentNames0[i], componentSt0[i]);
			componentSpt0Map.put(componentNames0[i], componentSpt0[i]);
			componentCpt0Map.put(componentNames0[i], componentCpt0[i]);
			componentODt0Map.put(componentNames0[i], componentODt0[i]);
		}

		Map observationOt0=new HashMap();
		String[]
flowRateSensorName0={"flowrateSensor1","flowrateSensor2","flowrateSensor3",
				"flowrateSensor4","flowrateSensor5","flowrateSensor6",
				"flowrateSensor7","flowrateSensor8","flowrateSensor9",
				"flowrateSensor10","flowrateSensor11","flowrateSensor12",
				"flowrateSensor13","flowrateSensor14","flowrateSensor17",
				"flowrateSensor18","flowrateSensor21"};
```

```
    String[] flowRateSensorNodes0={"f1","f2","f3",
                    "f4","f5","f6",
                    "f7","f8","f9",
                    "f10","f11","f12",
                    "f13","f14","f17",
                    "f18","f21"};
    for(int i=0;i<flowRateSensorName0.length;i++){
          observationOt0.put(flowRateSensorName0[i],  flowRateSensorNodes0[i]);
    }

    BNNetworkInfo[0].setBNName(BNNetworkAgentName[0]);
    BNNetworkInfo[0].setNodesName(nodesName0);
    BNNetworkInfo[0].setIntrac(intrac0);
    BNNetworkInfo[0].setConProbDis(conProbDis0);
    BNNetworkInfo[0].setNodesSize(nodeSize0);
    BNNetworkInfo[0].setNeighborsInfo(neighborsInfo0);
    BNNetworkInfo[0].setComponentCpt(componentCpt0Map);
    BNNetworkInfo[0].setComponentSpt(componentSpt0Map);
    BNNetworkInfo[0].setComponentSt(componentSt0Map);
    BNNetworkInfo[0].setComponentODt(componentODt0Map);
    BNNetworkInfo[0].setObservationOt(observationOt0);


    //service load1
Map conProbDis1=new HashMap();

    Map tableMapf14_1=new HashMap();
    String[] nodesName14_1={"f14"};
    tableMapf14_1.put("nodesName", nodesName14_1);
    double[] tempConProb14_1={0.5,0.5};
    tableMapf14_1.put("conProbDis",tempConProb14_1.clone());
    conProbDis1.put("f14", tableMapf14_1);

    Map tableMapf15=new HashMap();
    String[] nodesNamesf15={"St15","f14","f15"};
    tableMapf15.put("nodesName",nodesNamesf15);
    tableMapf15.put("conProbDis",ConditionalProbTypes.conProb10);
    conProbDis1.put("f15", tableMapf15);

    Map tableMapf16=new HashMap();
    String[] nodesNamesf16={"St16","f14","f16"};
    tableMapf16.put("nodesName",nodesNamesf16);
    tableMapf16.put("conProbDis",ConditionalProbTypes.conProb10);
    conProbDis1.put("f16", tableMapf16);

    Map tableMapf17_1=new HashMap();
    String[] nodesNamesf17_1={"f15","f16","f17"};
    tableMapf17_1.put("nodesName",nodesNamesf17_1);
    tableMapf17_1.put("conProbDis",ConditionalProbTypes.conProb13);
    conProbDis1.put("f17", tableMapf17_1);


    for(int i=0;i<nodesName1.length;i++){
           if(nodesName1[i].indexOf("Spt")!=-1){
                  Map tableMap=new HashMap();
                  String temp=nodesName1[i];
                  String[] nodesNameI={temp};
                  tableMap.put("nodesName",nodesNameI);
                  tableMap.put("conProbDis", ConditionalProbTypes.conProb1);
                  conProbDis1.put(nodesName1[i], tableMap);
           }else if(nodesName1[i].indexOf("Cpt")!=-1){
                  Map tableMap=new HashMap();
                  String temp=nodesName1[i];
                  String[] nodesNameI={temp};
                  tableMap.put("nodesName",nodesNameI);
                  tableMap.put("conProbDis", ConditionalProbTypes.conProb2);
                  conProbDis1.put(nodesName1[i], tableMap);
           }else if(nodesName1[i].indexOf("St")!=-1){
                  Map tableMap=new HashMap();
                  String temp1=nodesName1[i].substring(2);
                  String[] nodesNamesSt={"Spt"+temp1,"Cpt"+temp1,"St"+temp1};
```

```
                                tableMap.put("nodesName",nodesNamesSt);
                                tableMap.put("conProbDis", ConditionalProbTypes.conProb3);
                                conProbDis1.put(nodesName1[i], tableMap);
                        }else if(nodesName1[i].indexOf("ODt")!=-1){
                                Map tableMap=new HashMap();
                                String temp1=nodesName1[i].substring(3);
                                String[] nodesNamesSt={"St"+temp1,"ODt"+temp1};
                                tableMap.put("nodesName",nodesNamesSt);
                                tableMap.put("conProbDis", ConditionalProbTypes.conProb16);
                                conProbDis1.put(nodesName1[i], tableMap);
                        }
                }

                Map nodeSize1=new HashMap();
                for(int i=0;i<nodesName1.length;i++){
                        if(nodesName1[i].indexOf("f")!=-1){
                                nodeSize1.put(nodesName1[i], "2");
                        }else if((nodesName1[i].indexOf("St")!=-
1)||(nodesName1[i].indexOf("Spt")!=-1)){
                                nodeSize1.put(nodesName1[i], "4");
                        }else {
                                nodeSize1.put(nodesName1[i], "2");
                        }
                }

                Map neighborsInfo1=new HashMap();
                String[] shared10={"f14","f17"};
                neighborsInfo1.put(BNNetworkAgentName[0], shared10);

                Map componentSt1Map=new HashMap();
                Map componentSpt1Map=new HashMap();
                Map componentCpt1Map=new HashMap();
                Map componentODt1Map=new HashMap();

                String[]
componentNames1={"RSADSys_serviceload1_valve1","RSADSys_serviceload1_valve2"};

                String[] componentSt1={"St15","St16"};
                String[] componentSpt1={"Spt15","Spt16"};
                String[] componentCpt1={"Cpt15","Cpt16"};
                String[] componentODt1={"ODt15","ODt16"};

                for(int i=0;i<componentNames1.length;i++){
                        componentSt1Map.put(componentNames1[i], componentSt1[i]);
                        componentSpt1Map.put(componentNames1[i], componentSpt1[i]);
                        componentCpt1Map.put(componentNames1[i], componentCpt1[i]);
                        componentODt1Map.put(componentNames1[i], componentODt1[i]);
                }

                Map observationOt1=new HashMap();
                String[] flowRateSensorName1={"flowrateSensor14","flowrateSensor15",
                                "flowrateSensor16","flowrateSensor17"};

                String[] flowRateSensorNodes1={"f14","f15","f16","f17"};

                for(int i=0;i<flowRateSensorName1.length;i++){
                      observationOt1.put(flowRateSensorName1[i],  flowRateSensorNodes1[i]);
                }
                BNNetworkInfo[1].setBNName(BNNetworkAgentName[1]);
                BNNetworkInfo[1].setNodesName(nodesName1);
                BNNetworkInfo[1].setIntrac(intrac1);
                BNNetworkInfo[1].setConProbDis(conProbDis1);
                BNNetworkInfo[1].setNodesSize(nodeSize1);
                BNNetworkInfo[1].setNeighborsInfo(neighborsInfo1);
                BNNetworkInfo[1].setComponentCpt(componentCpt1Map);
                BNNetworkInfo[1].setComponentSpt(componentSpt1Map);
                BNNetworkInfo[1].setComponentSt(componentSt1Map);
                BNNetworkInfo[1].setComponentODt(componentODt1Map);
                BNNetworkInfo[1].setObservationOt(observationOt1);
```

```
		//service load 2
		Map conProbDis2=new HashMap();

		Map tableMapf18_2=new HashMap();
		String[] nodesNamesf18_2={"f18"};
		tableMapf18_2.put("nodesName",nodesNamesf18_2);
		double[] tempConProb18_2={0.5,0.5};
		tableMapf18_2.put("conProbDis",tempConProb18_2);
		conProbDis2.put("f18", tableMapf18_2);

		Map tableMapf19=new HashMap();
		String[] nodesNamesf19={"St18","f18","f19"};
		tableMapf19.put("nodesName",nodesNamesf19);
		tableMapf19.put("conProbDis",ConditionalProbTypes.conProb10);
		conProbDis2.put("f19", tableMapf19);

		Map tableMapf20=new HashMap();
		String[] nodesNamesf20={"St19","f18","f20"};
		tableMapf20.put("nodesName",nodesNamesf20);
		tableMapf20.put("conProbDis",ConditionalProbTypes.conProb10);
		conProbDis2.put("f20", tableMapf20);

		Map tableMapf21_2=new HashMap();
		String[] nodesNamesf21_2={"f19","f20","f21"};
		tableMapf21_2.put("nodesName",nodesNamesf21_2);
		tableMapf21_2.put("conProbDis",ConditionalProbTypes.conProb13);
		conProbDis2.put("f21", tableMapf21_2);


		for(int i=0;i<nodesName2.length;i++){
			if(nodesName2[i].indexOf("Spt")!=-1){
				Map tableMap=new HashMap();
				String temp=nodesName2[i];
				String[] nodesNameI={temp};
				tableMap.put("nodesName",nodesNameI);
				tableMap.put("conProbDis", ConditionalProbTypes.conProb1);
				conProbDis2.put(nodesName2[i], tableMap);
			}else if(nodesName2[i].indexOf("Cpt")!=-1){
				Map tableMap=new HashMap();
				String temp=nodesName2[i];
				String[] nodesNameI={temp};
				tableMap.put("nodesName",nodesNameI);
				tableMap.put("conProbDis", ConditionalProbTypes.conProb2);
				conProbDis2.put(nodesName2[i], tableMap);
			}else if(nodesName2[i].indexOf("St")!=-1){
				Map tableMap=new HashMap();
				String temp1=nodesName2[i].substring(2);
				String[] nodesNamesSt={"Spt"+temp1,"Cpt"+temp1,"St"+temp1};
				tableMap.put("nodesName",nodesNamesSt);
				tableMap.put("conProbDis", ConditionalProbTypes.conProb3);
				conProbDis2.put(nodesName2[i], tableMap);
			}else if(nodesName2[i].indexOf("ODt")!=-1){
				Map tableMap=new HashMap();
				String temp1=nodesName2[i].substring(3);
				String[] nodesNamesSt={"St"+temp1,"ODt"+temp1};
				tableMap.put("nodesName",nodesNamesSt);
				tableMap.put("conProbDis", ConditionalProbTypes.conProb16);
				conProbDis2.put(nodesName2[i], tableMap);
			}
		}

		Map nodeSize2=new HashMap();
		for(int i=0;i<nodesName2.length;i++){
			if(nodesName2[i].indexOf("f")!=-1){
				nodeSize2.put(nodesName2[i], "2");
			}else if((nodesName2[i].indexOf("St")!=-
1)||(nodesName2[i].indexOf("Spt")!=-1)){
				nodeSize2.put(nodesName2[i], "4");
			}else {
				nodeSize2.put(nodesName2[i], "2");
			}
```

```
                }

                Map neighborsInfo2=new HashMap();
                String[] shared20={"f18","f21"};
                neighborsInfo2.put(BNNetworkAgentName[0], shared20);

                Map componentSt2Map=new HashMap();
                Map componentSpt2Map=new HashMap();
                Map componentCpt2Map=new HashMap();
                Map componentODt2Map=new HashMap();

                String[]
componentNames2={"RSADSys_serviceload2_valve1","RSADSys_serviceload2_valve2"};

                String[] componentSt2={"St18","St19"};
                String[] componentSpt2={"Spt18","Spt19"};
                String[] componentCpt2={"Cpt18","Cpt19"};
                String[] componentODt2={"ODt18","ODt19"};

                for(int i=0;i<componentNames2.length;i++){
                        componentSt2Map.put(componentNames2[i], componentSt2[i]);
                        componentSpt2Map.put(componentNames2[i], componentSpt2[i]);
                        componentCpt2Map.put(componentNames2[i], componentCpt2[i]);
                        componentODt2Map.put(componentNames2[i], componentODt2[i]);
                }

                Map observationOt2=new HashMap();
                String[] flowRateSensorName2={"flowrateSensor18","flowrateSensor19",
                                "flowrateSensor20","flowrateSensor21"};

                String[] flowRateSensorNodes2={"f18","f19","f20","f21"};

                for(int i=0;i<flowRateSensorName2.length;i++){
                       observationOt2.put(flowRateSensorName2[i],  flowRateSensorNodes2[i]);
                }

                BNNetworkInfo[2].setBNName(BNNetworkAgentName[2]);
                BNNetworkInfo[2].setNodesName(nodesName2);
                BNNetworkInfo[2].setIntrac(intrac2);
                BNNetworkInfo[2].setConProbDis(conProbDis2);
                BNNetworkInfo[2].setNodesSize(nodeSize2);
                BNNetworkInfo[2].setNeighborsInfo(neighborsInfo2);
                BNNetworkInfo[2].setComponentCpt(componentCpt2Map);
                BNNetworkInfo[2].setComponentSpt(componentSpt2Map);
                BNNetworkInfo[2].setComponentSt(componentSt2Map);
                BNNetworkInfo[2].setComponentODt(componentODt2Map);
                BNNetworkInfo[2].setObservationOt(observationOt2);

                return BNNetworkInfo;

        }
}
```

```
package RSADsystem;

import java.io.IOException;
import java.util.*;


import component.*;
import msbn.util.*;

import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;
```

```
import Jama.*;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.table.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;

/**********************************************************************************************
***************
 * This is the whole ship chill water system agent.
 * It has the following functions.
 * Receive:
 * 1.   Information from its higher layer agent (such as priority of service
loads,resource capacities if the resources are not damaged at all etc)
 * 2.   Serviceload state and serviceload requirement from service load agents
 * 3.   Chill water resource states from chill water resource agents
 * 4.   Routes information from cross valve system agent
 * Send:
 * 1.   Reason the commands to its lower layer agents (such as service loads, chill water
resources, cross valve system) according to its lower layer system states and the
information from its higher layer agent.
 * 2.   Send some information required to its higher layer agent.
 * 3.   Initiate a GUI to show some important information and an interface for the system
manually controls the system.
 * @author Daili Zhang
 * @author Aerospace System Design Lab, Georgia Institute of Technology.
 * @version 1.0
 */

public class RSADSystemAgent extends Agent{

        private InfoFromShipLevel infoFromShipLevel=new InfoFromShipLevel(); //stores
information from ship level system.
        private double[] priority=infoFromShipLevel.getPriority(); //stores priorities of
service loads.
        private double[] serviceloadReq=new double[ConstantCollection.serviceloadNum];
//stores service load requirements.
        private double[] resourceCap=infoFromShipLevel.getResourceCap(); //stores chill
water resource capabilities.

        private InfoToShipLevel infoToShipLevel=new InfoToShipLevel(); //stores the
information which will be sent to ship level system.
        private ComponentState[] serviceloadState=infoToShipLevel.getServiceloadState();
//stores service load states.
        private ComponentState[] resourceState=infoToShipLevel.getResourceState();
//stores chill water resource states.
        private double[][] routesSummary=infoToShipLevel.getRoutesSummary(); //stores
general routes states for each service load.

        private Set serviceloadAgentName=new HashSet(); //stores service load agents'
names.
        private AID[] serviceloadAgent=new AID[ConstantCollection.serviceloadNum];
//stores service load agents IDs.
        private double[] serviceloadCommand=new double[ConstantCollection.serviceloadNum];
//stores service load commands which the second layer agent issues.
        private String[] serviceloadGetResourceFromWhichResource=new
String[ConstantCollection.serviceloadNum]; //R1 indicates from R1, R2 indicates from R2,
R0 indicates from nowhere.

        private Set resourceAgentName=new HashSet(); //stores chill water resource agents'
names.
        private AID[] resourceAgent=new AID[ConstantCollection.resourceNum]; //stores
chill water resource agents' IDs.
        private double[] resourceCommand=new double[ConstantCollection.resourceNum];
//stores chill water resource commands which the second layer agent issues.
```

```
        private AID crossValvesSystemAgent; //crossValves system agent's ID.
        private AID interfaceWithHighLevelAgent;

        private MessageTemplate mtFromServiceloadAgent; //filter out all messages except
messages from service load agents.
        private MessageTemplate mtFromResourceAgent; //filter out all messages except
messages from chill water resource agents.
        private MessageTemplate mtFromCrossValvesAgent; //filter out all messages except
messages from crossValves system agent.
        private MessageTemplate mtFromHighLevelInterfaceAgent; //filter out all messages
except messages from high level interface Agent.

        protected void setup(){
                for(int i=0;i<ConstantCollection.serviceloadNum;i++){
                        serviceloadAgentName.add(ConstantCollection.serviceloadNames[i]);
                        serviceloadAgent[i]=new
AID(ConstantCollection.serviceloadNames[i],AID.ISLOCALNAME);
                        serviceloadCommand[i]=0;
                        serviceloadGetResourceFromWhichResource[i]="R1";
                }
                for(int i=0;i<ConstantCollection.resourceNum;i++){

        resourceAgentName.add(ConstantCollection.chillWaterResourceNames[i]);
                        resourceAgent[i]=new
AID(ConstantCollection.chillWaterResourceNames[i],AID.ISLOCALNAME);
                        resourceCommand[i]=0;
                }

                crossValvesSystemAgent=new
AID(ConstantCollection.crossValvesSystemName,AID.ISLOCALNAME);
                interfaceWithHighLevelAgent=new
AID(ConstantCollection.interfaceWithShipLevelAgentName,AID.ISLOCALNAME);

                SenderNameMatchExpression matchSLD=new
SenderNameMatchExpression(serviceloadAgentName),
                matchRD=new SenderNameMatchExpression(resourceAgentName);
                mtFromServiceloadAgent=new MessageTemplate(matchSLD);
                mtFromResourceAgent=new MessageTemplate(matchRD);
                mtFromCrossValvesAgent=MessageTemplate.MatchSender(crossValvesSystemAgent);

        mtFromHighLevelInterfaceAgent=MessageTemplate.MatchSender(interfaceWithHighLevelAg
ent);

                addBehaviour(new AcceptInfoFromShipLevel(this));
                addBehaviour(new AcceptInfoFromServiceloadAgent(this));
                addBehaviour(new AcceptInfoFromResourceAgent(this));
                addBehaviour(new AcceptInfoFromCrossValvesAgent(this));
        }

        /**************************************************************************
*************************
         * This is an inner class accepting information from ship level agent
         */
        class AcceptInfoFromShipLevel extends CyclicBehaviour {
                AcceptInfoFromShipLevel(Agent a){
                        super(a);
                }

                public void action() {
                        ACLMessage msg=receive(mtFromHighLevelInterfaceAgent);
                        if(msg!=null){
                                try{

        infoFromShipLevel=(InfoFromShipLevel)msg.getContentObject();
                                }catch(UnreadableException ie0){
                                        ie0.printStackTrace();
                                }
                                double[] priorityN=infoFromShipLevel.getPriority();
                                double[] resourceCapN=infoFromShipLevel.getResourceCap();

                                boolean flag1=false,flag3=false;
```

```
                                for(int i=0;i<priorityN.length;i++){
                                        if(priorityN[i]!=priority[i]){
                                                flag1=true;
                                                break;
                                        }
                                }

                                for(int i=0;i<resourceCap.length;i++){
                                        if(resourceCapN[i]!=resourceCap[i]){
                                                flag3=true;
                                                break;
                                        }
                                }

                                if(flag1||flag3){
                                        priority=priorityN;
                                        resourceCap=resourceCapN;

                                        analyzeCommandToSubSystems();
                                        sendMessages();
                                }
                        }else{
                                block();
                        }
                }
        }

        /**************************************************************************************
**************************
         * This is an inner class accepting information from serviceload agent
         * "INFORM" for serviceload state
         * "PROPOGATE" for serviceload requirement
         */
        class AcceptInfoFromServiceloadAgent extends CyclicBehaviour{
                public  AcceptInfoFromServiceloadAgent(Agent a){
                        super(a);
                }
                public void action(){
                        ACLMessage msg=receive(mtFromServiceloadAgent);
                        if(msg!=null){
                                if(msg.getPerformative()==ACLMessage.INFORM){
                                        try{
                                                ComponentState
serviceS=(ComponentState)msg.getContentObject();
                                                String temp=msg.getSender().getLocalName();
                                                int i=temp.indexOf("serviceload");
                                                int
j=Integer.parseInt(temp.substring(i+"serviceload".length(),i+"serviceload".length()+1));
                                                serviceloadState[j-1]=serviceS;
                                        }catch(Exception e3)
                                        {
                                                e3.printStackTrace();
                                        }
                                }else if(msg.getPerformative()==ACLMessage.PROPAGATE){
                                        try{
                                                String serviceReqTemp=msg.getContent();
                                                double
serviceReqTempDouble=Double.parseDouble(serviceReqTemp);
                                                String temp=msg.getSender().getLocalName();
                                                int i=temp.indexOf("serviceload");
                                                int
j=Integer.parseInt(temp.substring(i+"serviceload".length(),i+"serviceload".length()+1));
                                                serviceloadReq[j-1]=serviceReqTempDouble;
                                        }catch(Exception e4){
                                                e4.printStackTrace();
                                        }
                                }
                                analyzeCommandToSubSystems();
                                sendMessages();
                        }else{
```

```
                                block();
                        }
                }
        }

        /**********************************************************************************
*************************
         * This is an inner class accepting information from resource center agent
         * "INFORM" for resource state
         */
        class AcceptInfoFromResourceAgent extends CyclicBehaviour{
                public  AcceptInfoFromResourceAgent(Agent a){
                        super(a);
                }
                public void action(){
                        ACLMessage msg=receive(mtFromResourceAgent);
                        if(msg!=null){
                                if(msg.getPerformative()==ACLMessage.INFORM){
                                        try{
                                                ComponentState
ResourceS=(ComponentState)msg.getContentObject();
                                                String temp=msg.getSender().getLocalName();
                                                int m=temp.indexOf("chillWaterResource");
                                                int
n=Integer.parseInt(temp.substring(m+"chillWaterResource".length(),m+"chillWaterResource".
length()+1));
                                                resourceState[n-1]=ResourceS;

                                                analyzeCommandToSubSystems();
                                                sendMessages();
                                        }catch(UnreadableException ie0){
                                                ie0.printStackTrace();
                                        }
                                }
                        }else{
                                block();
                        }
                }
        }


        /**********************************************************************************
*************************
         * This is an inner class accepting information from crossvalve system agent
         */
        class AcceptInfoFromCrossValvesAgent extends CyclicBehaviour{
                public  AcceptInfoFromCrossValvesAgent(Agent a){
                        super(a);
                }
                public void action(){
                        ACLMessage msg=receive(mtFromCrossValvesAgent);
                        if(msg!=null){
                                String routeS=msg.getContent();
                                String routeSN=routeS.replaceAll(" ", "");//It is important
that the state is split by ",".
                                String[] routeSTemp=routeSN.split(",");
                                int j=0;
                                for(int i=0;i<routeSTemp.length-1;){

        routesSummary[j][0]=Double.parseDouble(routeSTemp[i]);
                                        j++;
                                        i++;
                                        i++;
                                }
                                analyzeCommandToSubSystems();
                                sendMessages();
                        }else{
                                block();
                        }
                }
        }
```

```
        //*************************************************************************
        //Currently, this method is used for one resource center. For two resource center,
it needs to be modified.
        protected void analyzeCommandToSubSystems(){
                double[] comToR=new double[ConstantCollection.resourceNum];
                for(int i=0;i<comToR.length;i++){
                        comToR[i]=0;
                }

                double[] comToS=new double[ConstantCollection.serviceloadNum];
                String[] fromWhichResource=new String[ConstantCollection.serviceloadNum];
                for(int i=0;i<ConstantCollection.serviceloadNum;i++){
                        comToS[i]=0;
                        fromWhichResource[i]="R0";//can not get resource. R1 from resource
1, R2 from resource 2,R0 from nowhere.
                }

                double[] p=new double[priority.length];
                for(int kkk=0;kkk<priority.length;kkk++){
                        p[kkk]=priority[kkk];
                }

                double R1CLeft=resourceCap[0]*resourceState[0].getMaxOpenDegree();
                double R1Req=0;

                for(int m=0;m<p.length;m++){
                        int k=getMaxElementIndex(p);
                        p[k]=-1;

        if(routesSummary[k][0]==1&&R1CLeft>0&&serviceloadState[k].getMaxOpenDegree()>=0.1&
&serviceloadReq[k]!=0){
                                fromWhichResource[k]="R1";
                                if(R1CLeft>=serviceloadReq[k]){
                                        comToS[k]=serviceloadReq[k];
                                        R1CLeft=R1CLeft-serviceloadReq[k];
                                        R1Req=R1Req+serviceloadReq[k];
                                }else{
                                        comToS[k]=R1CLeft;
                                        R1Req=R1Req+R1CLeft;
                                        R1CLeft=0;
                                }
                        }
                        if(R1CLeft==0){
                                break;
                        }
                }

                if(R1Req>0){
                        comToR[0]=1;
                }

                /*
                int maxSLIndex=getMaxElementIndex(comToS);
                double maxSL=comToS[maxSLIndex];
                if(maxSL!=0){
                        for (int i=0;i<comToS.length;i++){
                                comToS[i]=comToS[i]/maxSL;
                        }
                }
                */
                serviceloadCommand=comToS;
                resourceCommand=comToR;
                serviceloadGetResourceFromWhichResource=fromWhichResource;
        }


        public int getMaxElementIndex(double[] a){
                int i=0;
                double max=-10000;
```

```
                for(int j=0;j<a.length;j++){
                        if(a[j]>max){
                                max=a[j];
                                i=j;
                        }
                }
                return i;
        }


        public void sendMessages(){
                infoToShipLevel.setServiceloadState(serviceloadState);
                infoToShipLevel.setResourceState(resourceState);
                infoToShipLevel.setRoutesSummary(routesSummary);

               ACLMessage  msgToSendInfoToShipLevelAgent=new  ACLMessage(ACLMessage.INFORM);
                try{
                        msgToSendInfoToShipLevelAgent.setContentObject(infoToShipLevel);

        msgToSendInfoToShipLevelAgent.addReceiver(interfaceWithHighLevelAgent);
                        send(msgToSendInfoToShipLevelAgent);
                }catch(IOException ie1){
                        ie1.printStackTrace();
                }

                ACLMessage msgToCrossValvesSys=new ACLMessage(ACLMessage.INFORM);
                String tempMsgToCrossValvesSys="  ";
                for(int i=0;i<(serviceloadGetResourceFromWhichResource.length)-1;i++){

        tempMsgToCrossValvesSys+=serviceloadGetResourceFromWhichResource[i]+",";
                }

        tempMsgToCrossValvesSys+=serviceloadGetResourceFromWhichResource[(serviceloadGetRe
sourceFromWhichResource.length)-1];
                msgToCrossValvesSys.setContent(tempMsgToCrossValvesSys);
                msgToCrossValvesSys.addReceiver(crossValvesSystemAgent);
                send(msgToCrossValvesSys);

                for(int ix=0;ix<ConstantCollection.serviceloadNum;ix++){
                        ACLMessage msgToServiceload=new ACLMessage(ACLMessage.INFORM);
                        msgToServiceload.setContent(new
Double(serviceloadCommand[ix]).toString());
                        msgToServiceload.addReceiver(serviceloadAgent[ix]);
                        send(msgToServiceload);
                }

                for(int ix=0;ix<ConstantCollection.resourceNum;ix++){
                        ACLMessage msgToResource=new ACLMessage(ACLMessage.INFORM);
                        msgToResource.setContent(new
Double(resourceCommand[ix]).toString());
                        msgToResource.addReceiver(resourceAgent[ix]);
                        send(msgToResource);
                }
        }
}
```

APPENDIX F

INTEGRATION AND SIMULATION RUN SCHEUDLER

All of the modules are integrated into ModelCenter and they are interacting with each other through inputs and outputs. There are 5 types of running mode in ModelCenter: Forward, Backward, Mixed, Parallel and Script. Compared with other modes, the Script scheduler mode is a special mode, in which components do not run automatically as fixed. Instead, a user-supplied script is invoked that tells which components to run and in which order to run them.

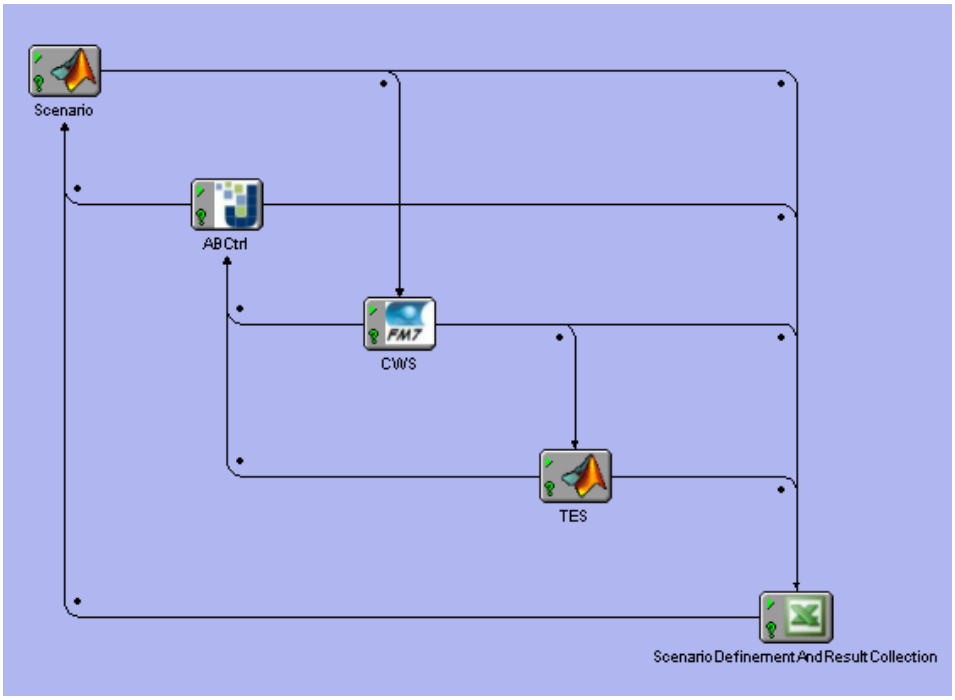


FIGURE 216 MULT-AGENT BASED CONTROL WITH DYNAMIC INFERENCE ENGINE FOR CHILLED WATER SYSTEM SIMULATION ENVIRONMENT

The script can be written in several script languages: ECMAScript, JavaScript, JScript, LiveScript, SignedJavaScript, SignedVBScript, VBS, VBScript and XML. In the application of this dissertation, the script is written in VBScript as shown in the following. It reads scenarios from an Excel file named ScenarioDefinementAndResultCollection.xls , runs different modules shown in Figure 216 coordinately and collects data back into the Excel file automatically. In the script, a few ideas are borrowed from my colleague Matt Hoepfer. After the simulation finishes, the Excel worksheet is as shown in Figure 217.

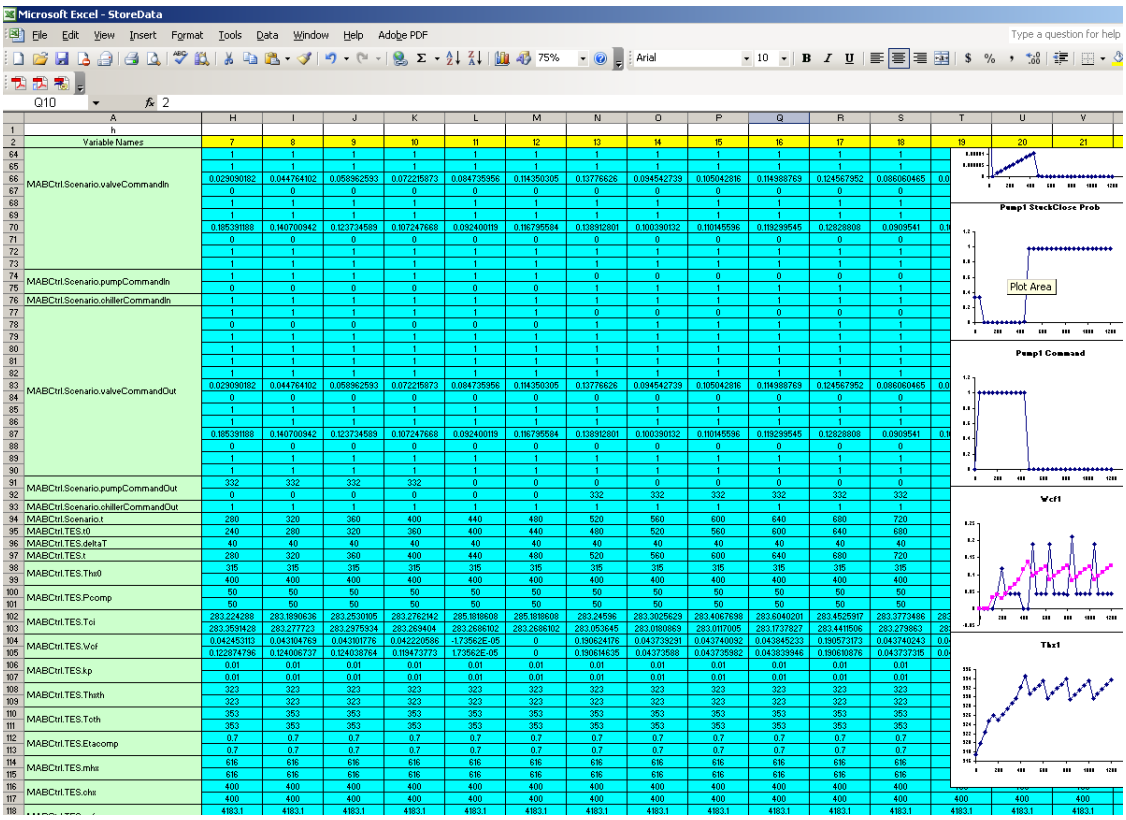


FIGURE 217 SCREEN SHOT OF SCENARIODEFINEMENTANDRESULTCOLLECTION WORKSHEET

```
'-----------------------------------------------------------------------------------------------
option explicit

dim CWS,ABCtrl,TES,Scenario
dim t0,deltaT
dim valve6Com,valve7Com,valve10Com,valve11Com
```

```
dim valve6OD,valve7OD,valve10OD,valve11OD
dim f14,f15,f18,f19
dim f13,f17
dim FBIterations
dim CWSDeltaT
dim i
dim countGlobalIteration,totalGlobalIterations

'Use an Excel file to store results
dim Excel,ExcelOn,ExcelWorkbook,ExcelWorksheet,selCell

'Count colum number and row number and array size
dim countRows, countCols, countArrayLen

'String is used to store names of variables
dim varName,sharedVarNamePart,varObj,varNameSpec


sub run

set CWS=app.getComponent("MABCtrl.CWS")
set ABCtrl=app.getComponent("MABCtrl.ABCtrl")
set TES=app.getComponent("MABCtrl.TES")
set Scenario=app.getComponent("MABCtrl.Scenario")

CountGlobalIteration=0
totalGlobalIterations=30


'Choose storing results or not
ExcelOn=0
if MsgBox("Display Excel?",4,"Excel")=6 then
ExcelOn=1
end if

'if storing data is chosen, open an existed Excel file and initialize it
if ExcelOn=1 then
set Excel=CreateObject("Excel.Application")
Excel.Visible=True
set
ExcelWorkbook=Excel.WorkBooks.open("C:\Data\Research\AboutThesis\Integration\StoreData.xl
s")
ExcelWorkbook.Worksheets("Result1").Activate
Excel.Range("B60:IV400").Select
Excel.Selection.ClearContents
Excel.Selection.Font.Size=10
Excel.Selection.WrapText = True
Excel.Selection.Borders.Weight=2
Excel.Selection.HorizontalAlignment = 3
Excel.Selection.VerticalAlignment = 2
Excel.Selection.Unmerge
Excel.Range("A1").Select


'Initialize number of rows, number of columns
countRows=0
countCols=0
countArrayLen=0

'Write variables' names to first row of the opened Excel file
countCols=countCols+1
countRows=countRows+3

'*****************************************************************************
'Start collecting variables' names to Excel file
'*****************************************************************************
'****************Start collecting Scenario variable names*******************
sharedVarNamePart="MABCtrl.Scenario."

varName=sharedVarNamePart&"t0"
Excel.Cells(countRows,countCols)=varName
```

```
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"deltaT"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"valveState"
set varObj=Scenario.getVariable("valveState")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge

countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpState"
set varObj=Scenario.getVariable("pumpState")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerState"
set varObj=Scenario.getVariable("chillerState")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

sharedVarNamePart="MABCtrl.ABCtrl."

varName=sharedVarNamePart&"sensedFlowRate"
set varObj=ABCtrl.getVariable("sensedFlowRate")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"sensedValveOpenDegree"
set varObj=ABCtrl.getVariable("sensedValveOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"sensedPumpOpenDegree"
set varObj=ABCtrl.getVariable("sensedPumpOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"sensedChillerOpenDegree"
set varObj=ABCtrl.getVariable("sensedChillerOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
```

```
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen


sharedVarNamePart="MABCtrl.Scenario."

varName=sharedVarNamePart&"valveCommandIn"
set varObj=Scenario.getVariable("valveCommandIn")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpCommandIn"
set varObj=Scenario.getVariable("pumpCommandIn")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerCommandIn"
set varObj=Scenario.getVariable("chillerCommandIn")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveCommandOut"
set varObj=Scenario.getVariable("valveCommandOut")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpCommandOut"
set varObj=Scenario.getVariable("pumpCommandOut")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerCommandOut"
set varObj=Scenario.getVariable("chillerCommandOut")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"t"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

'****************End collecting Scenario variable names*******************
'****************Start collecting TES variable names*******************
sharedVarNamePart="MABCtrl.TES."

varName=sharedVarNamePart&"t0"
```

```
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"deltaT"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"t"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"Thx0"
set varObj=TES.getVariable("Thx0")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Pcomp"
set varObj=TES.getVariable("Pcomp")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Tci"
set varObj=TES.getVariable("Tci")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Wcf"
set varObj=TES.getVariable("Wcf")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"kp"
set varObj=TES.getVariable("kp")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Thxth"
set varObj=TES.getVariable("Thxth")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Tcth"
set varObj=TES.getVariable("Tcth")
countArrayLen=varObj.length
```

```
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Etacomp"
set varObj=TES.getVariable("Etacomp")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"mhx"
set varObj=TES.getVariable("mhx")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chx"
set varObj=TES.getVariable("chx")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"ccf"
set varObj=TES.getVariable("ccf")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"A"
set varObj=TES.getVariable("A")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"h"
set varObj=TES.getVariable("h")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Tco"
set varObj=TES.getVariable("Tco")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Thx"
set varObj=TES.getVariable("Thx")
```

```
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"Wcfdesire"
set varObj=TES.getVariable("Wcfdesire")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

'***************End collecting TES variable names*****************************

'***************Start collecting ABCtrl variable names**********************
sharedVarNamePart="MABCtrl.ABCtrl."

varName=sharedVarNamePart&"t0"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"deltaT"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"t"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"chillerCloseProb"
set varObj=ABCtrl.getVariable("chillerCloseProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerCommand"
set varObj=ABCtrl.getVariable("chillerCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerOpenDegree"
set varObj=ABCtrl.getVariable("chillerOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerOpenProb"
set varObj=ABCtrl.getVariable("chillerOpenProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen
```

```
varName=sharedVarNamePart&"chillerStuckCloseProb"
set varObj=ABCtrl.getVariable("chillerStuckCloseProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerStuckOpenProb"
set varObj=ABCtrl.getVariable("chillerStuckOpenProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"flowrate"
set varObj=ABCtrl.getVariable("flowrate")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpCloseProb"
set varObj=ABCtrl.getVariable("pumpCloseProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpCommand"
set varObj=ABCtrl.getVariable("pumpCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpOpenDegree"
set varObj=ABCtrl.getVariable("pumpOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpOpenProb"
set varObj=ABCtrl.getVariable("pumpOpenProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpStuckCloseProb"
set varObj=ABCtrl.getVariable("pumpStuckCloseProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
```

```
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpStuckOpenProb"
set varObj=ABCtrl.getVariable("pumpStuckOpenProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"resourceCapacity"
set varObj=ABCtrl.getVariable("resourceCapacity")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"serviceloadPriority"
set varObj=ABCtrl.getVariable("serviceloadPriority")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"serviceloadReq"
set varObj=ABCtrl.getVariable("serviceloadReq")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveCloseProb"
set varObj=ABCtrl.getVariable("valveCloseProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveCommand"
set varObj=ABCtrl.getVariable("valveCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveOpenDegree"
set varObj=ABCtrl.getVariable("valveOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveOpenProb"
set varObj=ABCtrl.getVariable("valveOpenProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
```

```
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveStuckCloseProb"
set varObj=ABCtrl.getVariable("valveStuckCloseProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveStuckOpenProb"
set varObj=ABCtrl.getVariable("valveStuckOpenProb")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

'***************End collecting ABCtrl variable names*************************

'***************Start collecting CWS variable names*************************
sharedVarNamePart="MABCtrl.CWS."

varName=sharedVarNamePart&"t0"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"deltaT"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"t"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"timeStep"
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
countRows=countRows+1

varName=sharedVarNamePart&"valveCommand"
set varObj=CWS.getVariable("valveCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"ruptureCommand"
set varObj=CWS.getVariable("ruptureCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpCommand"
set varObj=CWS.getVariable("pumpCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
```

```
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerValveCommand"
set varObj=CWS.getVariable("chillerValveCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerToutCommand"
set varObj=CWS.getVariable("chillerToutCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"SLToutCommand"
set varObj=CWS.getVariable("SLToutCommand")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"flowmeter"
set varObj=CWS.getVariable("flowmeter")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"SLTin"
set varObj=CWS.getVariable("SLTin")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"valveOpenDegree"
set varObj=CWS.getVariable("valveOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"ruptureOpenDegree"
set varObj=CWS.getVariable("ruptureOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"pumpSpeed"
set varObj=CWS.getVariable("pumpSpeed")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
```

```
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen

varName=sharedVarNamePart&"chillerValveOpenDegree"
set varObj=CWS.getVariable("chillerValveOpenDegree")
countArrayLen=varObj.length
Excel.Cells(countRows,countCols)=varName
Excel.Cells(countRows,countCols).name=varName
Excel.Range(Excel.Cells(countRows,countCols),Excel.Cells(countRows+countArrayLen-
1,countCols)).Merge
countRows=countRows+countArrayLen
'****************End collecting CWS variable names*************************
'**************************************************************************
'End collecting variables' names to Excel file
'**************************************************************************
'**************************************************************************
'start running integration model for totalGlobalIterations times
'**************************************************************************

for countGlobalIteration=0 to totalGlobalIterations
'**************************************************************************
'Start reading Scenario data and Sensor state data from Excel file
'**************************************************************************
sharedVarNamePart="MABCtrl.Scenario."

varName=sharedVarNamePart&"t0"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
app.setValue varName,selCell.value


varName=sharedVarNamePart&"deltaT"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
app.setValue varName,selCell.value

varName=sharedVarNamePart&"valveState"
set varObj=Scenario.getVariable("valveState")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
next

varName=sharedVarNamePart&"pumpState"
set varObj=Scenario.getVariable("pumpState")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
next

varName=sharedVarNamePart&"chillerState"
set varObj=Scenario.getVariable("chillerState")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
next

sharedVarNamePart="MABCtrl.ABCtrl."
varName=sharedVarNamePart&"sensedFlowRate"
set varObj=ABCtrl.getVariable("sensedFlowRate")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
```

```
next

varName=sharedVarNamePart&"sensedValveOpenDegree"
set varObj=ABCtrl.getVariable("sensedValveOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
next

varName=sharedVarNamePart&"sensedPumpOpenDegree"
set varObj=ABCtrl.getVariable("sensedPumpOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
next

varName=sharedVarNamePart&"sensedChillerOpenDegree"
set varObj=ABCtrl.getVariable("sensedChillerOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
app.setValue varNameSpec,selCell.offset(i,0).value
next
'*****************************************************************************
'End reading Scenario data and Sensor state data from Excel file
'*****************************************************************************
'*****************************************************************************
'Now need to run the whole model for one time
'*****************************************************************************
Scenario.run
t0=app.getValue("MABCtrl.Scenario.t0")
deltaT=app.getValue("MABCtrl.Scenario.deltaT")

FBIterations=80
CWSDeltaT=deltaT/FBIterations

app.setValue "MABCtrl.ABCtrl.t0",t0
app.setValue "MABCtrl.ABCtrl.deltaT",deltaT
ABCtrl.run

app.setValue  "MABCtrl.Scenario.valveCommandIn",app.getValue("MABCtrl.ABCtrl.valveCommand")
app.setValue "MABCtrl.Scenario.pumpCommandIn",app.getValue("MABCtrl.ABCtrl.pumpCommand")
app.setValue
"MABCtrl.Scenario.chillerCommandIn",app.getValue("MABCtrl.ABCtrl.chillerCommand")
Scenario.run

app.setValue "MABCtrl.CWS.valveCommand",app.getValue("MABCtrl.Scenario.valveCommandOut")
app.setValue "MABCtrl.CWS.pumpCommand",app.getValue("MABCtrl.Scenario.pumpCommandOut")
app.setValue
"MABCtrl.CWS.chillerValveCommand",app.getValue("MABCtrl.Scenario.chillerCommandOut")

valve6Com=app.getValue("MABCtrl.Scenario.valveCommandOut[6]")
valve7Com=app.getValue("MABCtrl.Scenario.valveCommandOut[7]")
valve10Com=app.getValue("MABCtrl.Scenario.valveCommandOut[10]")
valve11Com=app.getValue("MABCtrl.Scenario.valveCommandOut[11]")

dim CWSt0
CWSt0=t0

for i=1 to FBIterations
   app.setValue "MABCtrl.CWS.t0", CWSt0

   valve6OD=app.getValue("MABCtrl.CWS.valveOpenDegree[6]")
   valve7OD=app.getValue("MABCtrl.CWS.valveOpenDegree[7]")
   valve10OD=app.getValue("MABCtrl.CWS.valveOpenDegree[10]")
   valve11OD=app.getValue("MABCtrl.CWS.valveOpenDegree[11]")
```

```
f14=app.getValue("MABCtrl.CWS.flowmeter[14]")
f15=app.getValue("MABCtrl.CWS.flowmeter[15]")
f18=app.getValue("MABCtrl.CWS.flowmeter[18]")
f19=app.getValue("MABCtrl.CWS.flowmeter[19]")


if(abs(f14-valve6Com)>0.00001) then
   if (f14<valve6Com) then
      valve6OD=valve6OD+1/FBIterations
   else
      valve6OD=valve6OD-1/FBIterations
   end if
end if
if(valve6OD>1) then valve6OD=1
if(valve6OD<0) then valve6OD=0

if(valve6OD<=0) then
   if(valve6Com>0) then
     valve6OD=1/FBIterations
   end if
end if

if(valve6Com<=0) then valve6OD=0

app.setValue "MABCtrl.CWS.valveCommand[6]",valve6OD


if(abs(f15-valve7Com)>0.00001) then
   if (f15<valve7Com) then
      valve7OD=valve7OD+1/FBIterations
   else
      valve7OD=valve7OD-1/FBIterations
   end if
end if
if(valve7OD>1) then valve7OD=1
if(valve7OD<=0) then valve7OD=0

if(valve7OD<=0) then
   if(valve7Com>0) then
     valve7OD=1/FBIterations
   end if
end if

 if(valve7Com<=0) then valve7OD=0

app.setValue "MABCtrl.CWS.valveCommand[7]",valve7OD

if(abs(f18-valve10Com)>0.00001) then
   if (f18<valve10Com) then
      valve10OD=valve10OD+1/FBIterations
   else
      valve10OD=valve10OD-1/FBIterations
   end if
end if
if(valve10OD>1) then valve10OD=1
if(valve10OD<=0) then valve10OD=0

if(valve10OD<=0) then
   if(valve10Com>0) then
     valve10OD=1/FBIterations
   end if
end if

 if(valve10Com<=0) then valve10OD=0

app.setValue "MABCtrl.CWS.valveCommand[10]",valve10OD

if(abs(f19-valve11Com)>0.00001) then
   if (f19<valve11Com) then
      valve11OD=valve11OD+1/FBIterations
```

```
      else
         valve11OD=valve11OD-1/FBIterations
      end if
   end if
   if(valve11OD>1) then valve11OD=1
   if(valve11OD<=0) then valve11OD=0

   if(valve11OD<=0) then
      if(valve11Com>0) then
        valve11OD=1/FBIterations
      end if
   end if

    if(valve11Com<=0) then valve11OD=0

   app.setValue "MABCtrl.CWS.valveCommand[11]",valve11OD

   app.setValue "MABCtrl.CWS.deltaT",CWSDeltaT
   CWSt0=CWSt0+CWSDeltaT
   CWS.run
next

   app.setValue "MABCtrl.TES.t0",t0
   app.setValue "MABCtrl.TES.deltaT",deltaT

   f13=app.getValue("MABCtrl.CWS.flowmeter[13]")
   f17=app.getValue("MABCtrl.CWS.flowmeter[17]")
   app.setValue "MABCtrl.TES.Wcf[0]",f13
   app.setValue "MABCtrl.TES.Wcf[1]",f17
   app.setValue "MABCtrl.TES.Tci",app.getValue("MABCtrl.CWS.SLTin")

   TES.run

'*******************************************************************************
'Finished running the whole model for one time
'*******************************************************************************
'*******************************************************************************
'Writing Data to Excel file
'*******************************************************************************
'Start writing Scenario variables' values to Excel file
sharedVarNamePart="MABCtrl.Scenario."

varName=sharedVarNamePart&"valveCommandIn"
set varObj=Scenario.getVariable("valveCommandIn")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpCommandIn"
set varObj=Scenario.getVariable("pumpCommandIn")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerCommandIn"
set varObj=Scenario.getVariable("chillerCommandIn")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveCommandOut"
set varObj=Scenario.getVariable("valveCommandOut")
```

```
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpCommandOut"
set varObj=Scenario.getVariable("pumpCommandOut")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerCommandOut"
set varObj=Scenario.getVariable("chillerCommandOut")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"t"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

'*************End writing Scenario variables' value to Excel file
'**********Start writing TES variables' value to Excel file
sharedVarNamePart="MABCtrl.TES."

varName=sharedVarNamePart&"t0"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"deltaT"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"t"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"Thx0"
set varObj=TES.getVariable("Thx0")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Pcomp"
set varObj=TES.getVariable("Pcomp")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Tci"
set varObj=TES.getVariable("Tci")
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next
```

```
varName=sharedVarNamePart&"Wcf"
set varObj=TES.getVariable("Wcf")
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"kp"
set varObj=TES.getVariable("kp")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Thxth"
set varObj=TES.getVariable("Thxth")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Tcth"
set varObj=TES.getVariable("Tcth")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Etacomp"
set varObj=TES.getVariable("Etacomp")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"mhx"
set varObj=TES.getVariable("mhx")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chx"
set varObj=TES.getVariable("chx")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"ccf"
set varObj=TES.getVariable("ccf")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next
```

```
varName=sharedVarNamePart&"A"
set varObj=TES.getVariable("A")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"h"
set varObj=TES.getVariable("h")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Tco"
set varObj=TES.getVariable("Tco")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Thx"
set varObj=TES.getVariable("Thx")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"Wcfdesire"
set varObj=TES.getVariable("Wcfdesire")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

'*************End writing TES variables' value to Excel file
'*************Start writing ABCtrl variables' values to Excel file
sharedVarNamePart="MABCtrl.ABCtrl."

varName=sharedVarNamePart&"t0"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"deltaT"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"t"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"chillerCloseProb"
set varObj=ABCtrl.getVariable("chillerCloseProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerCommand"
```

```
set varObj=ABCtrl.getVariable("chillerCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerOpenDegree"
set varObj=ABCtrl.getVariable("chillerOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerOpenProb"
set varObj=ABCtrl.getVariable("chillerOpenProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerStuckCloseProb"
set varObj=ABCtrl.getVariable("chillerStuckCloseProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerStuckOpenProb"
set varObj=ABCtrl.getVariable("chillerStuckOpenProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"flowrate"
set varObj=ABCtrl.getVariable("flowrate")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpCloseProb"
set varObj=ABCtrl.getVariable("pumpCloseProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpCommand"
set varObj=ABCtrl.getVariable("pumpCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next
```

```
varName=sharedVarNamePart&"pumpOpenDegree"
set varObj=ABCtrl.getVariable("pumpOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpOpenProb"
set varObj=ABCtrl.getVariable("pumpOpenProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpStuckCloseProb"
set varObj=ABCtrl.getVariable("pumpStuckCloseProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpStuckOpenProb"
set varObj=ABCtrl.getVariable("pumpStuckOpenProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"resourceCapacity"
set varObj=ABCtrl.getVariable("resourceCapacity")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next


varName=sharedVarNamePart&"serviceloadPriority"
set varObj=ABCtrl.getVariable("serviceloadPriority")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"serviceloadReq"
set varObj=ABCtrl.getVariable("serviceloadReq")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveCloseProb"
set varObj=ABCtrl.getVariable("valveCloseProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
```

```
next

varName=sharedVarNamePart&"valveCommand"
set varObj=ABCtrl.getVariable("valveCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveOpenDegree"
set varObj=ABCtrl.getVariable("valveOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveOpenProb"
set varObj=ABCtrl.getVariable("valveOpenProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveStuckCloseProb"
set varObj=ABCtrl.getVariable("valveStuckCloseProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveStuckOpenProb"
set varObj=ABCtrl.getVariable("valveStuckOpenProb")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

'****************End writing ABCtrl variables' values to Excel file
'****************Start writing CWS variables' values to Excel file
sharedVarNamePart="MABCtrl.CWS."

varName=sharedVarNamePart&"t0"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"deltaT"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"t"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"timeStep"
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
selCell.value=app.getValue(varName)

varName=sharedVarNamePart&"valveCommand"
set varObj=CWS.getVariable("valveCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
```

```
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"ruptureCommand"
set varObj=CWS.getVariable("ruptureCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next
varName=sharedVarNamePart&"pumpCommand"
set varObj=CWS.getVariable("pumpCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerValveCommand"
set varObj=CWS.getVariable("chillerValveCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerToutCommand"
set varObj=CWS.getVariable("chillerToutCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"SLToutCommand"
set varObj=CWS.getVariable("SLToutCommand")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"flowmeter"
set varObj=CWS.getVariable("flowmeter")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"SLTin"
set varObj=CWS.getVariable("SLTin")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"valveOpenDegree"
set varObj=CWS.getVariable("valveOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
```

```
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"ruptureOpenDegree"
set varObj=CWS.getVariable("ruptureOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"pumpSpeed"
set varObj=CWS.getVariable("pumpSpeed")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next

varName=sharedVarNamePart&"chillerValveOpenDegree"
set varObj=CWS.getVariable("chillerValveOpenDegree")
countArrayLen=varObj.length
set selCell=Excel.Range(varName).offset(0,1+countGlobalIteration)
for i=0 to countArrayLen-1
varNameSpec=varName&"["&i&"]"
selCell.offset(i,0)=app.getValue(varNameSpec)
next
'End writing CWS variables' values to Excel file
'End writing data to Excel file
app.setValue "MABCtrl.ABCtrl.flowrate",app.getValue("MABCtrl.CWS.flowmeter")
 app.setValue  "MABCtrl.ABCtrl.valveOpenDegree",app.getValue("MABCtrl.CWS.valveOpenDegree")
 app.setValue "MABCtrl.ABCtrl.pumpOpenDegree",app.getValue("MABCtrl.CWS.pumpSpeed")
 app.setValue
"MABCtrl.ABCtrl.chillerOpenDegree",app.getValue("MABCtrl.CWS.chillerValveOpenDegree")
 app.setValue "MABCtrl.ABCtrl.serviceloadReq",app.getValue("MABCtrl.TES.Wcfdesire")
 app.setValue "MABCtrl.ABCtrl.serviceloadPriority",app.getValue("MABCtrl.TES.Priority")
next

'Save and close the opened Excel file
Excel.DisplayAlerts=False
ExcelWorkbook.Save()
Excel.Quit

end if

end sub
```

APPENDIX G

PRIOR AND CONDITIONAL DISTRIBUTIONS USED IN THE APPLICATION

FULLBNT has the capability of taking potentials as variable distributions. In the application, all of the prior and conditional distributions are represented as potentials and listed in the following tables. Nodes names and positions are refered to Figure 88.

S_i^{t-1}	C_i^{t-1}	S_i^t	$P\left(S_i^t \mid S_i^{t-1}, C_i^{t-1}\right)$
Open	Open	Open	1
Close	Open	Open	1
StuckOpen	Open	Open	0.000001
StuckClose	Open	Open	0.000001
Open	Close	Open	0.000001
Close	Close	Open	0.000001
StuckOpen	Close	Open	0.000001
StuckClose	Close	Open	0.000001
Open	Open	Close	0.000001
Close	Open	Close	0.000001
StuckOpen	Open	Close	0.000001
StuckClose	Open	Close	0.000001
Open	Close	Close	1
Close	Close	Close	1
StuckOpen	Close	Close	0.000001
StuckClose	Close	Close	0.000001
Open	Open	StuckOpen	0.000001
Close	Open	StuckOpen	0.000001
StuckOpen	Open	StuckOpen	1
StuckClose	Open	StuckOpen	0.000001
Open	Close	StuckOpen	0.000001
Close	Close	StuckOpen	0.000001
StuckOpen	Close	StuckOpen	1
StuckClose	Close	StuckOpen	0.000001
Open	Open	StuckClose	0.000001
Close	Open	StuckClose	0.000001
StuckOpen	Open	StuckClose	0.000001
StuckClose	Open	StuckClose	1
Open	Close	StuckClose	0.000001
Close	Close	StuckClose	0.000001
StuckOpen	Close	StuckClose	0.000001
StuckClose	Close	StuckClose	1

S_1^t	f_9	f_{13}	f_1	$P(f_1 \mid S_1^t, f_9, f_{13})$
S_7^t	f_5	f_6	f_7	$P(f_7 \mid S_7^t, f_5, f_6)$
Open	NoFlow	NoFlow	NoFlow	1
Close	NoFlow	NoFlow	NoFlow	1
StuckOpen	NoFlow	NoFlow	NoFlow	1
StuckClose	NoFlow	NoFlow	NoFlow	1
Open	PositiveFlow	NoFlow	NoFlow	0.000001
Close	PositiveFlow	NoFlow	NoFlow	0.5
StuckOpen	PositiveFlow	NoFlow	NoFlow	0.000001
StuckClose	PositiveFlow	NoFlow	NoFlow	0.5
Open	NoFlow	PositiveFlow	NoFlow	0.000001
Close	NoFlow	PositiveFlow	NoFlow	0.5
StuckOpen	NoFlow	PositiveFlow	NoFlow	0.000001
StuckClose	NoFlow	PositiveFlow	NoFlow	0.5
Open	PositiveFlow	PositiveFlow	NoFlow	0.000001
Close	PositiveFlow	PositiveFlow	NoFlow	0.5
StuckOpen	PositiveFlow	PositiveFlow	NoFlow	0.000001
StuckClose	PositiveFlow	PositiveFlow	NoFlow	0.5
Open	NoFlow	NoFlow	PositiveFlow	0.000001
Close	NoFlow	NoFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	NoFlow	PositiveFlow	0.000001
StuckClose	NoFlow	NoFlow	PositiveFlow	0.000001
Open	PositiveFlow	NoFlow	PositiveFlow	1
Close	PositiveFlow	NoFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	NoFlow	PositiveFlow	1
StuckClose	PositiveFlow	NoFlow	PositiveFlow	0.5
Open	NoFlow	PositiveFlow	PositiveFlow	1
Close	NoFlow	PositiveFlow	PositiveFlow	0.5
StuckOpen	NoFlow	PositiveFlow	PositiveFlow	1
StuckClose	NoFlow	PositiveFlow	PositiveFlow	0.5
Open	PositiveFlow	PositiveFlow	PositiveFlow	1
Close	PositiveFlow	PositiveFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	PositiveFlow	PositiveFlow	1
StuckClose	PositiveFlow	PositiveFlow	PositiveFlow	0.5

$S_i^{t_0}$	$P(S_i^{t_0})$
Open	0.25
Close	0.25
StuckOpen	0.25
StuckClose	0.25

$C_i^{t_0}$	$P(C_i^{t_0})$
Open	0.5
Close	0.5

S_2^t	f_2	$P(f_2 \mid S_2^t)$
Open	NoFlow	0.5
Close	NoFlow	1
StuckOpen	NoFlow	0.5
StuckClose	NoFlow	1
Open	PositiveFlow	0.5
Close	PositiveFlow	0.000001
StuckOpen	PositiveFlow	0.5
StuckClose	PositiveFlow	0.000001

S_i^t	OD_i^t	$P(OD_i^t \mid S_i^t)$
Open	Open	1
Close	Open	0.000001
StuckOpen	Open	1
StuckClose	Open	0.000001
Open	Close	0.000001
Close	Close	1
StuckOpen	Close	0.000001
StuckClose	Close	1

S_{10}^{t}	f_8	f_{12}	f_{10}	$P\left(f_{10} \mid S_{10}^{t}, f_8, f_{12}\right)$
S_{11}^{t}	f_{17}	f_{21}	f_{11}	$P\left(f_{11} \mid S_{11}^{t}, f_{17}, f_{21}\right)$
Open	NoFlow	NoFlow	NegativeFlow	0.000001
Close	NoFlow	NoFlow	NegativeFlow	0.000001
StuckOpen	NoFlow	NoFlow	NegativeFlow	0.000001
StuckClose	NoFlow	NoFlow	NegativeFlow	0.000001
Open	PositiveFlow	NoFlow	NegativeFlow	0.000001
Close	PositiveFlow	NoFlow	NegativeFlow	0.000001
StuckOpen	PositiveFlow	NoFlow	NegativeFlow	0.000001
StuckClose	PositiveFlow	NoFlow	NegativeFlow	0.000001
Open	NoFlow	PositiveFlow	NegativeFlow	0.5
Close	NoFlow	PositiveFlow	NegativeFlow	0.000001
StuckOpen	NoFlow	PositiveFlow	NegativeFlow	0.5
StuckClose	NoFlow	PositiveFlow	NegativeFlow	0.000001
Open	PositiveFlow	PositiveFlow	NegativeFlow	0.3333333
Close	PositiveFlow	PositiveFlow	NegativeFlow	0.000001
StuckOpen	PositiveFlow	PositiveFlow	NegativeFlow	0.3333333
StuckClose	PositiveFlow	PositiveFlow	NegativeFlow	0.000001
Open	NoFlow	NoFlow	NoFlow	1
Close	NoFlow	NoFlow	NoFlow	1
StuckOpen	NoFlow	NoFlow	NoFlow	1
StuckClose	NoFlow	NoFlow	NoFlow	1
Open	PositiveFlow	NoFlow	NoFlow	0.5
Close	PositiveFlow	NoFlow	NoFlow	1
StuckOpen	PositiveFlow	NoFlow	NoFlow	0.5
StuckClose	PositiveFlow	NoFlow	NoFlow	1
Open	NoFlow	PositiveFlow	NoFlow	0.5
Close	NoFlow	PositiveFlow	NoFlow	1
StuckOpen	NoFlow	PositiveFlow	NoFlow	0.5
StuckClose	NoFlow	PositiveFlow	NoFlow	1
Open	PositiveFlow	PositiveFlow	NoFlow	0.3333333
Close	PositiveFlow	PositiveFlow	NoFlow	1
StuckOpen	PositiveFlow	PositiveFlow	NoFlow	0.3333333
StuckClose	PositiveFlow	PositiveFlow	NoFlow	1
Open	NoFlow	NoFlow	PositiveFlow	0.000001
Close	NoFlow	NoFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	NoFlow	PositiveFlow	0.000001
StuckClose	NoFlow	NoFlow	PositiveFlow	0.000001
Open	PositiveFlow	NoFlow	PositiveFlow	0.5
Close	PositiveFlow	NoFlow	PositiveFlow	0.000001
StuckOpen	PositiveFlow	NoFlow	PositiveFlow	0.5
StuckClose	PositiveFlow	NoFlow	PositiveFlow	0.000001
Open	NoFlow	PositiveFlow	PositiveFlow	0.000001
Close	NoFlow	PositiveFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	PositiveFlow	PositiveFlow	0.000001
StuckClose	NoFlow	PositiveFlow	PositiveFlow	0.000001
Open	PositiveFlow	PositiveFlow	PositiveFlow	0.3333333
Close	PositiveFlow	PositiveFlow	PositiveFlow	0.000001
StuckOpen	PositiveFlow	PositiveFlow	PositiveFlow	0.3333333
StuckClose	PositiveFlow	PositiveFlow	PositiveFlow	0.000001

S_{13}^{t}	f_{21}	f_{11}	f_{13}	$P\left(f_{13} \mid S_{13}^{t}, f_{11}, f_{13}\right)$
Open	NoFlow	NegativeFlow	NoFlow	0.5
Close	NoFlow	NegativeFlow	NoFlow	0.5
StuckOpen	NoFlow	NegativeFlow	NoFlow	0.5
StuckClose	NoFlow	NegativeFlow	NoFlow	0.5
Open	PositiveFlow	NegativeFlow	NoFlow	0.000001
Close	PositiveFlow	NegativeFlow	NoFlow	1
StuckOpen	PositiveFlow	NegativeFlow	NoFlow	0.000001
StuckClose	PositiveFlow	NegativeFlow	NoFlow	1
Open	NoFlow	NoFlow	NoFlow	0.5
Close	NoFlow	NoFlow	NoFlow	1
StuckOpen	NoFlow	NoFlow	NoFlow	1
StuckClose	NoFlow	NoFlow	NoFlow	1
Open	PositiveFlow	NoFlow	NoFlow	0.000001
Close	PositiveFlow	NoFlow	NoFlow	0.5
StuckOpen	PositiveFlow	NoFlow	NoFlow	0.000001
StuckClose	PositiveFlow	NoFlow	NoFlow	0.5
Open	NoFlow	PositiveFlow	NoFlow	0.000001
Close	NoFlow	PositiveFlow	NoFlow	0.5
StuckOpen	NoFlow	PositiveFlow	NoFlow	0.000001
StuckClose	NoFlow	PositiveFlow	NoFlow	0.5
Open	PositiveFlow	PositiveFlow	NoFlow	0.000001
Close	PositiveFlow	PositiveFlow	NoFlow	0.5
StuckOpen	PositiveFlow	PositiveFlow	NoFlow	0.000001
StuckClose	PositiveFlow	PositiveFlow	NoFlow	0.5
Open	NoFlow	NegativeFlow	PositiveFlow	0.5
Close	NoFlow	NegativeFlow	PositiveFlow	0.5
StuckOpen	NoFlow	NegativeFlow	PositiveFlow	0.5
StuckClose	NoFlow	NegativeFlow	PositiveFlow	0.5
Open	PositiveFlow	NegativeFlow	PositiveFlow	1
Close	PositiveFlow	NegativeFlow	PositiveFlow	0.000001
StuckOpen	PositiveFlow	NegativeFlow	PositiveFlow	1
StuckClose	PositiveFlow	NegativeFlow	PositiveFlow	0.000001
Open	NoFlow	NoFlow	PositiveFlow	0.5
Close	NoFlow	NoFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	NoFlow	PositiveFlow	0.000001
StuckClose	NoFlow	NoFlow	PositiveFlow	0.000001
Open	PositiveFlow	NoFlow	PositiveFlow	1
Close	PositiveFlow	NoFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	NoFlow	PositiveFlow	1
StuckClose	PositiveFlow	NoFlow	PositiveFlow	0.5
Open	NoFlow	PositiveFlow	PositiveFlow	1
Close	NoFlow	PositiveFlow	PositiveFlow	0.5
StuckOpen	NoFlow	PositiveFlow	PositiveFlow	1
StuckClose	NoFlow	PositiveFlow	PositiveFlow	0.5
Open	PositiveFlow	PositiveFlow	PositiveFlow	1
Close	PositiveFlow	PositiveFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	PositiveFlow	PositiveFlow	1
StuckClose	PositiveFlow	PositiveFlow	PositiveFlow	0.5

S_9^t	f_{17}	f_{11}	f_9	$P(f_9 \mid S_9^t, f_{17}, f_{11})$
Open	NoFlow	NegativeFlow	NoFlow	0.000001
Close	NoFlow	NegativeFlow	NoFlow	0.5
StuckOpen	NoFlow	NegativeFlow	NoFlow	0.000001
StuckClose	NoFlow	NegativeFlow	NoFlow	0.5
Open	PositiveFlow	NegativeFlow	NoFlow	0.000001
Close	PositiveFlow	NegativeFlow	NoFlow	0.5
StuckOpen	PositiveFlow	NegativeFlow	NoFlow	0.000001
StuckClose	PositiveFlow	NegativeFlow	NoFlow	0.5
Open	NoFlow	NoFlow	NoFlow	1
Close	NoFlow	NoFlow	NoFlow	1
StuckOpen	NoFlow	NoFlow	NoFlow	1
StuckClose	NoFlow	NoFlow	NoFlow	1
Open	PositiveFlow	NoFlow	NoFlow	0.000001
Close	PositiveFlow	NoFlow	NoFlow	0.5
StuckOpen	PositiveFlow	NoFlow	NoFlow	0.000001
StuckClose	PositiveFlow	NoFlow	NoFlow	0.5
Open	NoFlow	PositiveFlow	NoFlow	0.5
Close	NoFlow	PositiveFlow	NoFlow	0.5
StuckOpen	NoFlow	PositiveFlow	NoFlow	0.5
StuckClose	NoFlow	PositiveFlow	NoFlow	0.5
Open	PositiveFlow	PositiveFlow	NoFlow	0.000001
Close	PositiveFlow	PositiveFlow	NoFlow	1
StuckOpen	PositiveFlow	PositiveFlow	NoFlow	0.000001
StuckClose	PositiveFlow	PositiveFlow	NoFlow	1
Open	NoFlow	NegativeFlow	PositiveFlow	1
Close	NoFlow	NegativeFlow	PositiveFlow	0.5
StuckOpen	NoFlow	NegativeFlow	PositiveFlow	1
StuckClose	NoFlow	NegativeFlow	PositiveFlow	0.5
Open	PositiveFlow	NegativeFlow	PositiveFlow	1
Close	PositiveFlow	NegativeFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	NegativeFlow	PositiveFlow	1
StuckClose	PositiveFlow	NegativeFlow	PositiveFlow	0.5
Open	NoFlow	NoFlow	PositiveFlow	0.000001
Close	NoFlow	NoFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	NoFlow	PositiveFlow	0.000001
StuckClose	NoFlow	NoFlow	PositiveFlow	0.000001
Open	PositiveFlow	NoFlow	PositiveFlow	1
Close	PositiveFlow	NoFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	NoFlow	PositiveFlow	1
StuckClose	PositiveFlow	NoFlow	PositiveFlow	0.5
Open	NoFlow	PositiveFlow	PositiveFlow	0.5
Close	NoFlow	PositiveFlow	PositiveFlow	0.5
StuckOpen	NoFlow	PositiveFlow	PositiveFlow	0.5
StuckClose	NoFlow	PositiveFlow	PositiveFlow	0.5
Open	PositiveFlow	PositiveFlow	PositiveFlow	1
Close	PositiveFlow	PositiveFlow	PositiveFlow	0.000001
StuckOpen	PositiveFlow	PositiveFlow	PositiveFlow	1
StuckClose	PositiveFlow	PositiveFlow	PositiveFlow	0.000001

S_5^t	f_3	f_5	$P\left(f_5 \mid S_5^t, f_3\right)$
S_6^t	f_4	f_6	$P\left(f_6 \mid S_6^t, f_4\right)$
Open	NoFlow	NoFlow	1
Close	NoFlow	NoFlow	1
StuckOpen	NoFlow	NoFlow	1
StuckClose	NoFlow	NoFlow	1
Open	PositiveFlow	NoFlow	0.000001
Close	PositiveFlow	NoFlow	0.5
StuckOpen	PositiveFlow	NoFlow	0.000001
StuckClose	PositiveFlow	NoFlow	0.5
Open	NoFlow	PositiveFlow	0.000001
Close	NoFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	PositiveFlow	0.000001
StuckClose	NoFlow	PositiveFlow	0.000001
Open	PositiveFlow	PositiveFlow	1
Close	PositiveFlow	PositiveFlow	0.5
StuckOpen	PositiveFlow	PositiveFlow	1
StuckClose	PositiveFlow	PositiveFlow	0.5

S_3^t	f_2	f_3	$P\left(f_3 \mid S_3^t, f_2\right)$
S_4^t	f_2	f_4	$P\left(f_4 \mid S_4^t, f_2\right)$
S_8^t	f_7	f_8	$P\left(f_8 \mid S_8^t, f_7\right)$
S_{12}^t	f_7	f_{12}	$P\left(f_{12} \mid S_{12}^t, f_7\right)$
S_{15}^t	f_{14}	f_{15}	$P\left(f_{15} \mid S_{15}^t, f_{14}\right)$
S_{16}^t	f_{14}	f_{16}	$P\left(f_{16} \mid S_{16}^t, f_{14}\right)$
S_{18}^t	f_{18}	f_{19}	$P\left(f_{19} \mid S_{18}^t, f_{18}\right)$
S_{19}^t	f_{18}	f_{20}	$P\left(f_{20} \mid S_{19}^t, f_{19}\right)$
StuckOpen	NoFlow	NoFlow	1
StuckClose	NoFlow	NoFlow	1
Open	PositiveFlow	NoFlow	0.5
Close	PositiveFlow	NoFlow	1
StuckOpen	PositiveFlow	NoFlow	0.5
StuckClose	PositiveFlow	NoFlow	1
Open	NoFlow	PositiveFlow	0.000001
Close	NoFlow	PositiveFlow	0.000001
StuckOpen	NoFlow	PositiveFlow	0.000001
StuckClose	NoFlow	PositiveFlow	0.000001
Open	PositiveFlow	PositiveFlow	0.5
Close	PositiveFlow	PositiveFlow	0.000001
StuckOpen	PositiveFlow	PositiveFlow	0.5
StuckClose	PositiveFlow	PositiveFlow	0.000001

f_{12}	f_{10}	f_{18}	$P(f_{18} \mid f_{12}, f_{10})$
NoFlow	NegativeFlow	NoFlow	0.5
PositiveFlow	NegativeFlow	NoFlow	0.5
NoFlow	NoFlow	NoFlow	1
PositiveFlow	NoFlow	NoFlow	0.000001
NoFlow	PositiveFlow	NoFlow	0.000001
PositiveFlow	PositiveFlow	NoFlow	0.000001
NoFlow	NegativeFlow	PositiveFlow	0.5
PositiveFlow	NegativeFlow	PositiveFlow	0.5
NoFlow	NoFlow	PositiveFlow	0.000001
PositiveFlow	NoFlow	PositiveFlow	1
NoFlow	PositiveFlow	PositiveFlow	1
PositiveFlow	PositiveFlow	PositiveFlow	1

f_8	f_{10}	f_{14}	$P(f_{14} \mid f_8, f_{10})$
NoFlow	NegativeFlow	NoFlow	0.000001
PositiveFlow	NegativeFlow	NoFlow	0.000001
NoFlow	NoFlow	NoFlow	1
PositiveFlow	NoFlow	NoFlow	0.000001
NoFlow	PositiveFlow	NoFlow	0.5
PositiveFlow	PositiveFlow	NoFlow	0.5
NoFlow	NegativeFlow	PositiveFlow	1
PositiveFlow	NegativeFlow	PositiveFlow	1
NoFlow	NoFlow	PositiveFlow	0.000001
PositiveFlow	NoFlow	PositiveFlow	1
NoFlow	PositiveFlow	PositiveFlow	0.5
PositiveFlow	PositiveFlow	PositiveFlow	0.5

f_{15}	f_{16}	f_{17}	$P(f_{17} \mid f_{15}, f_{16})$
f_{19}	f_{20}	f_{21}	$P(f_{21} \mid f_{19}, f_{20})$
NoFlow	NoFlow	NoFlow	1
PositiveFlow	NoFlow	NoFlow	0.000001
NoFlow	PositiveFlow	NoFlow	0.000001
PositiveFlow	PositiveFlow	NoFlow	0.000001
NoFlow	NoFlow	PositiveFlow	0.000001
PositiveFlow	NoFlow	PositiveFlow	1
NoFlow	PositiveFlow	PositiveFlow	1
PositiveFlow	PositiveFlow	PositiveFlow	1

REFERENCES

[1] "http://en.wikipedia.org/wiki/Complex_system." Accessed on Jun 2009.

[2] J. M. Talent, E. M. Kennedy, R. G. Bartlett, and G. Taylor, "Progress of the DD(X) Destroyer Program," *United States Government Accountability Office,* 2005.

[3] G. Vahtsevanos, L. Tang, and J. Reimann, "An Intelligent Approach to Coordinated Control of Multiple Unmanned Aerial Vehicles," *American Helicopter Society 60th Annual Forum, Baltimore, MD, USA,* 2004.

[4] J. Lewe, "An Integrated Decision-Making Framework for Transportation Architectures: Application to Aviation Systems Design," *PHD Dissertation, Georgia Institute of Technology,* 2005.

[5] "http://www.globalsecurity.org/military/systems/ship/systems.htm." Accessed on Dec 2009.

[6] J. Jung, "Intelligent Systems for Strategic Power Infrastructure Defense," *phD Dissertation, University of Washington,* 2002.

[7] A. S. Rao and M. P. Georgeff, "BDI Agents: from Theory to Practice," *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95),* 1995.

[8] H. A. Simon, "The Science of the Artificial," *MIT Press Science/ Philosphy,* 1996.

[9] N. R. Jennings and S. Bussmann, "Agent-Based Control Systems: Why Are They Suited to Engineering Complex Systems?," *Control Systems Magazine, IEEE,* vol. 23, pp. 61-73, 2003.

[10] J. Kurien, "Model-Based Monitoring, Diagnosis and Control," *PhD Thesis Proposal, Brown University Department of Computer Science,* 2003.

[11] D. P. Buse and Q. H. Wu, *IP Network-Based Multi-Agent Systems for Industrial Automation : Information Management, Condition Monitoring and Control of Power Systems*. London: Springer, 2007.

[12] V. R. Lesser and D. D. Corkill, "Functionally Accurate, Cooperative Distributed Systems," *IEEE Trans. on Systems, Man and Cybernetics, SMC-11,* 1981.

[13] V. R. Lesser and D. D. Corkill, "Cooperative Distributed Problem Solving: A New Approach for Structuring Distributed Systems," *Technical Report 78-7, Department of Computer and Information Science, University of massachusetts,* 1978.

[14] L. Dunnington, H. Stevens, and G. Grater, "Integrated Engineering Plant for Future Naval Combatants - Technology Assessment and Demonstration Roadmap," *technical Report Systems Engineering Group, Engineering Technology Center, Marine Technology Division,* 2003.

[15] C. B. Ruckman and D. B. Ward, "Auxiliary Electrical System Control via High-Speed Communication Networks," *Technical Papers,* 2006.

[16] B. Pierre, *Mathematical Principles of Signal Processing : Fourier and Wavelet Analysis*. New York: Springer, 2002.

[17] F. L. Lewis, "Wireless Sensor Networks," *Smart Environments: Technologies, Protocols, and Application,* 2004.

[18] W. V. D. Hoek and M. Wooldridge, "Towards A Logic of Rational Agency," *Logic J. IGPL, Oxford University Press,* vol. 11, pp. 135-160, 2003.

[19] D. H. Scheidt, "Intelligent Agent-Based Control," *Johns Hopkins APL Technical Digest,* vol. 23, 2002.

[20] M. Bratman, "Intentions, Plans and Practical Reason," *Harvard University Press,* 1987.

[21] F. Ingrand, M. Georgeff, and A. Rao, "An Architeture for Real-Time Reasoning and System Control," *IEEE Expert,* 1992.

[22] D. H. Scheidt and M. J. Pekala, "Model-Based Agents," *Power Engineering Society General Meeting, IEEE,* 2007.

[23] P. Vrba and V. Marik, "Simulation in Agent-Based Manufacturing Control Systems," *Systems, Man and Cybernetics, International Conference, IEEE,* 2005.

[24] H. V. D. Parunak, "A Practitioners' Review of Industrial Agent Applications," *Autonmous Agents and Multi-Agent Systems,* 2000.

[25] F. Karray, O. Basir, I. Song, and H. Li, "A Framework for Coordinated Control of Multi-Agent Systems," in *Intelligent Control, 2004. Proceedings of the 2004 IEEE International Symposium*, 2004, pp. 156-161.

[26] K. Fregene, D. Kennedy, and D. Wang, "HICA: A Framework for Distributed Multiagent Control," *IASTED International Conference on Intelligent Control,* pp. 187-192, 2001.

[27] F. F. Wu, "Parallel Processing in Power Systems Computation," *IEEE Trans. on PWRS,* vol. 7, 1992.

[28] J. B. Carvalho and F. M. Barbosa, "Parallel and Distributed Processing in State Estimation of Power System Energy," in *Electrotechnical Conference, 1998. MELECON 98., 9th Mediterranean*, 1998, pp. 969-973 vol.2.

[29] J. B. Carvalho and F. M. Barbosa, "Distributed Processing in Power System State Estimation," in *Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean*, 2000, pp. 1128-1131 vol.3.

[30] I. Romanovski and P. E. Caines, "On the Supervisory Control of Multi-Agent Product Systems," *Preceedings of the 41st IEEE, Conference on Decision and Control,* 2002.

[31] D. Naso and B. Turchiano, "A Coordination Strategy for Distributed Multi-Agent Manufacturing Systems," *Iranian Journal of Physics Research (IJPR),* 2004.

[32] FIPA, "FIPA abstract architecture specification," *Foundation for Intelligent Physical Agents,* 2002.

[33] F. Maturana, R. Staron, F. Discenzo, D. Scheidt, M. Pekala, J. Bracy, and M. Zink, "An Interoperable Agent-Based Control System for Survivable Shipboard Automation," *Proceedings of ASNE Reconfiguration and Survivability,* 2005.

[34] G. Booch, "Object-Oriented Analysis and Design with Applications," *Readings, MA: Addison Wesley,* 1994.

[35] F. P. Brooks, "The Mythical Man-Month," *Reading, MA: Addison Wesley,* 1995.

[36] R. R. Negenborn, B. D. Schutter, and J. Hellendoorn, "Multi-Agent Model Predictive Control: A Survey," *Delft Center for Systems and Control, Delft University of Technology,* 2004.

[37] S. Munroe and M. Luck, "Agent Autonomy Through the 3M Motivational Taxonomy," *Agents and Computational Autonomy,* 2004.

[38] B. Eckel, "Thiniking in Java," *3rd Edition,* 2003.

[39] M. Schillo and K. Fischer, "A Taxonomy of Autonomy in Multiagent Organization," *Agents and Computational Autonomy,* 2004.

[40] F. L. Bellifemine, G. Caire, and D. Greenwood, "Developing Multi-Agent System with JADE," *Wiley Series in Agent Technology,* 2004.

[41] D. Greenwood, "The Foundation for Intelligent, Physical Agents," *Whitestein Technologies AG,* 2004.

[42] G. Caire, "JADE Tutorial: JADE Programmer for Beginners," *TILAB,* 2003.

[43] "FIPA Request Interaction Protocol Specification," *Foundation for Intelligent Physical Agents,* 2002.

[44] D. Heckerman, "A Tutorial on Learning with Bayesian Networks," *Technical Report, Microsoft Research, Advanced Technology Division, Microsoft Corporation,* 1996.

[45] J. V. Finn, *Bayesian Networks and Decision Graphs*. New York: Springer, 2001.

[46] G. Cooper, "Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks," *Artificial Intelligence,* 1990.

[47] P. Dagum and M. Luby, "Approximating Probabilistic Inference in Bayesian Belief Networks Is Np-hard," *Artificial Intelligence,* 1993.

[48] L. Getoor and B. Taskar, "Introduction to Statistical Relational Learning," *MIT Press/Relational Database,* 2007.

[49] A. Pfeffer and T. Tai, "Asynchronous Dynamic Bayesian Networks," *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence,* 2005.

[50] "http://en.wikipedia.org/wiki/Inference_engine." Accessed on Jan 2008.

[51] L. Harrison, "http://www.aiai.ed.ac.uk/links/cbr.html#intro," 1997. Accessed on Jun 2009.

[52] "http://www.emeraldinsight.com/fig/1570140202001.png." Accessed on May 2009.

[53] N. L. Griffin and F. D. Lewis, "A Rule-Based Inference Engine Which Is Optimal and VLSI lmplementable," *Tools for Artificial Intelligence, Architectures, Languages and Algorithms, IEEE International Workshop,* 1989.

[54] "http://en.wikipedia.org/wiki/Forward_chaining." Accessed on Sep 2009.

[55] "http://en.wikipedia.org/wiki/Backward_chaining." Accessed on Sep 2009.

[56] C. L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," in *Expert Systems: A Software Methodology for Modern Applications*: IEEE Computer Society Press, 1990, pp. 324-341.

[57] F. D. McKenzie, A. J. Gonzalez, and R. Morris, "An Integrated Model-Based Approach for Real-Time On-Line Diagnosis of Complex Systems," *Engineering Applications of Artificial Intelligence,* 1998.

[58] B. J. Rodriguez and W. F. S. Poehlman, "A System for Distributed Inference," *Rochester 4th Conference,* 1996.

[59] M. M. Perianu and P. Havinga, "D-FLER: A Distributed Fuzzy Logic Engine for Rule-based Wireless Sensor Networks," *Ubiquitous Computing Systems, Lecture Notes in Computer Science,* 2007.

[60] W. Maul, C. Meyer, A. Jankovsky, and C. Fulton, "Qualitative Model-Based Diagnostics for Rocket Systems," *29th Joint Propulsion Conference, Lewis Research Center, National Aeronautics and Space Administration,* 1993.

[61] B. C. Williams and P. P. Nayak, "A Model-Based Approach to Reactive Self-Configuring Systems," *Proceedings of AAAI,* 1996.

[62] Y. Karp, M. H. Lee, and C. J. Price, "A Model-Based Diagnostic Engine for Electro-Mechanical Devices," *Department of Computer Science, University of Wales Aberystwyth,* 2003.

[63] V. Lessor, C. Ortiz, and M. Tambe, "Distributed Sensor Networks: A Multiagent Perspective (Edited Book)," *Series: Multiagent Systems, Artificial Societies, and Simulated Organizations. Kluwer Academic Publishers, Boston Hardbound,* 2003.

[64] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides, "A Survey on Sensor Networks from a Multi-Agent Perspective," *A workshop of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-08),* 2008.

[65] I. Sitharama, S. and R. Brooks, R., "Distributed Sensor Networks," *Chapman & Hall/CRC Computer and Information Science Series,* 2004.

[66] R. Olfati-Saber, "Distributed Kalman Filtering and Sensor Fusion in Sensor Networks," *Workshop on Networked Embedded Sensing and Control, University of Notre Dame,* 2005.

[67] R. Olfati-Saber, "Distributed Kalman Filter with Embedded Consensus Filters," *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conferenc,* 2005.

[68] P. J. Lawrence and Jr., "Comparison of a Distributed Kalman Filter Versus a Centralized Kalman Filter with Fault Detection Considerations," *Master thesis, Air Force Inst of Tech Wright-Patterson AFB OH School of Engineering,* 1993.

[69] J. P. Lawrence, Jr., and M. P. Berarducci, "Navigation Sensor, Filter, and Failure Mode Simulation Results Using the Distributed Kalman Filter Simulator (DKFSIM)," *Position Location and Navigation Symposium,* 1996.

[70] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, "Distributed Kalman Filtering Using Consensus Strategies," *Decision and Control, 2007 46th IEEE Conference,* 2007.

[71] S. Kirti and A. Scaglione, "Scalable Distributed Kalman Filtering through Consensus," *Acoustics, Speech and Signal Processing, ICASSP, IEEE International Conference,* 2008.

[72] M. Coates, "Distributed Particle Filters for Sensor Networks," *Proc. of 3nd Workshop on Information Processing in Sensor Networks,* 2004.

[73] M. Rosencrantz, G. Gordon, and S. Thrun, "Decentralized Sensor Fusion with Distributed Particle Filters," *The Proc. Conf. Uncertainty in Artificial Intelligence,* 2003.

[74] M. Bolic, P. M. Djuric, and S. Hong, "Resampling Algorithms and Architectures for Distributed Particle Filters," *Signal Processing, IEEE Transactions,* 2005.

[75] D. Gu, J. Sun, Z. Hu, and H. Li, "Consensus Based Distributed Particle Filter in Sensor Networks," *Information and Automation, ICIA, IEEE International Conference,* 2008.

[76] Q. Cheng and P. K. Varshney, "Joint State Monitoring and Fault Detection Using Distributed Particle Filtering," *Signals, Systems and Computers,* 2007.

[77] S. Funiak, C. Guestrin, M. Paskin, and S. R., "Distributed Inference in Dynamical Systems," *Advances in Neural Information Processing Systems,* 2006.

[78] P. d. Oude, "Belief Propagation in Distributed Probabilistic Models: Comparing Different Approaches to Probabilistic Inference in Distributed Bayesian Networks," *Master Thesis,* 2006.

[79] Y. Xiang, "Distributed Multi-Agent Probabilistic Reasoning with Bayesian Networks," *Methodologies for Intelligent Systems,* 1994.

[80] M. A. Paskin and C. E. Guestrin, "Robust Probabilistic Inference in Distributed Systems," *In Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence, AUAI Press,* pp. 436-445, 2004.

[81] R. Olfati-Saber and J. S. Shamma, "Consensus Filters for Sensor Networks and Distributed Sensor Fusion," *Preceedings of the 44th Conference on Decision and Control,* 2005.

[82] D. Spanos, R. Olfati-Saber, and R. M. Murray, "Dynamic Consensus on Mobile Networks," *The 16th IFAC World Congress,* 2005.

[83] B. Anderson and J. Moore, "Optimal Filtering," *Prentice-Hall, New Jersey,* 1979.

[84] R. Bucy and K. Senne, "Digital Synthesis of Nonlinear Filters," *Autmatica,* 1971.

[85] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich, "Collaborative Signal and Information Processing: An Information Directed Approach," *Proceedings of the IEEE,* 2003.

[86] L. Doherty, B. A. Warneke, B. E. Boser, and K. S. J. Pister, "Energy and Performance Considerations for Smart Dust," *International Journal of Parallel Distributed Systems and Networks,* 2001.

[87] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring Network Security Using Dynamic Bayesian Network," *Conference on Computer and Communications Security. Proceedings of the 4th ACM Workshop on Quality of Protection,* 2008.

[88] G. Zweig and S. Russell, "Speech Recognition with Dynamic Bayesian Networks," *AAAI-98 Proceedings,* 1998.

[89] H. Kao, C. Huang, and H. Li, "Supply Chain Diagnostics with Dynamic Bayesian Networks," *Computers and Industrial Engineering,* 2005.

[90] P. Sebastiani, K. D. Mandl, P. Szolovits, I. S. Kohane, and M. F. Ramoni, "A Bayesian Dynamic Model for Influenza Surveilance," *Statistics in Medicine,* 2006.

[91] K. P. Murphy, "Dynamic Baysian Networks," *Probabilistic Graphical Models,* 2002.

[92] M. A. Paskin, "Exploiting Locality in Probabilistic Inference," *PhD Dissertation,* 2004.

[93] K. P. Murphy, "Dynamic Bayesian Networks: Representation, Inference and Learning," *PhD Dissertation, University of California, Berkeley,* 2002.

[94] M. Schmidt and K. P. Murphy, "Modeling Discrete Interventional Data Using Directed Cyclic Graphical Models," *Conference on Uncertainty in Artificial Intelligence,* 2009.

[95] J. Pearl, "Prababilistic Reasoning in Intelligent Systems: Networks of Plausible Inference," *Morgan Kaufmann,* 1988.

[96] Y. Xiang and V. Lesser, "On the Role of Multiply Sectioned Bayesian Networks to Cooperative Multiagent Systems," *Systems, Man and Cybernetics, Part A, IEEE Transactions on,* vol. 33, pp. 489-501, 2003.

[97] G. Pavlin, P. d. Oude, and M. Maris, "Distributed Perception Networks: An Architecture for Information Fusion Systems Based on Causal Probabilistic Models," *Multisensor Fusion and Integration for Intelligent Systems, IEEE International Conference,* 2006.

[98] Y. Xiang, D. Poole, and M. P. Beddoes, "Multiply Sectioned Bayesian Networks and Junction Forests for Large Knowlege Based Systems," *Computational Intelligence,* vol. 19, 1993.

[99] Y. Xiang, "Distributed Multi-Agent Probabilistic Reasoning with Bayesian Networks," *Methodologies for Intelligent Systems,* 1994.

[100] Y. Xiang, "Multi-Agent Distributed Interpretation with Multiply Sectioned Bayesian Networks," *International Workshop on Multi-Agent Systems, MIT,* 1997.

[101] Y. Xiang, "Semantics of Multiply Sectioned Bayesian Networks for Cooperative Multi-agent Distributed Interpretation," *Advances in Artificial Intelligence,* 1996.

[102] Y. Xiang, "Cooperative Trianglulation in MSBNs without Revealing Subnet Structures," *Networks,* vol. 37, 2001.

[103] Y. Xiang, "Belief Updating in Multiply Sectioned Bayesian Networks without Repeated Local Propagations," *International Journal of Approximate Reasoning,* vol. 23, 2000.

[104] Y. Xiang, K. G. Olesen, and F. V. Jensen, "Practical Issues in Modeling Large Diagnostic Systems with Multiply Sectioned Bayesian Networks," *International Journal of Pattern Reconginition and Artificial Intelligence,* vol. 14, 2000.

[105] Y. Xiang and K. Zhang, "Agent Interface Enhancement: Making Multiagent Graphical Models Accessible," *Proc. 5th Inter. Joint Conf. on Autonomous Agents and Multiagent Systems,* 2006.

[106] E. Charniak, "Bayesian Networks without Tears," *AI Magazine,* 1991.

[107] "http://www.ics.uci.edu/~eppstein/161/960206.html." Accessed on May 2009.

[108] R. G. Gallager, P. A. Humblet, and P. M. Spira, "A Distributed Algorithm for Minimum-Weight Spanning Trees," *ACM Transactions on Programming Languages and Systems,* vol. 5, 1983.

[109] B. Awerbuch, "Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and Related Problems," *Proceedings of the 19th ACM Symposium on Theory of Computing (STOC),* 1987.

[110] J. Garay, S. Kutten, and D. Peleg, "A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Tree," *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS),* 1993.

[111] Y. Xiang, "Cooperative Verification of Agent Interface," *Probabilistic Graphical Models,* 2002.

[112] "System Specificationk, Airbus A330/A340 Flight Control System," *System Engineering,* 2001.

[113] N. Motee and B. Sayyar-Rodsari, "Optimal Partitioning in Distributed Model Predictive Control," *Proceedings of 2003 American Control Conference,* 2003.

[114] M. R. Katebi and M. A. Johnson, "Predictive Control Design for Large-Scale Systems," *Automatica,* 1997.

[115] M. W. Braun, D. E. Rivera, M. E. Flores, W. M. Carlyle, and K. G. Kempf, "A Model Predictive Control Framework for Robust Management of Multi-Product, Multi-Chelon Demand Networks," *Annual Reviews in Control,* 2003.

[116] H. E. Fawal, D. Georges, and G. Bornard, "Optimal Control of Complex Irrigation Systems via Decomposition-Coordination and the Use of Augmented

Lagrangian," *Proceedings of 1998 IEEE International on Systems, Man, and Cybernetics,* 1998.

[117] G. Georges, "Decentralized Adaptive Control for A Water Distribution System," *Proceedings of the 3rd IEEE Conference on Control Applications,* 1999.

[118] T. Roeller, "Smart Valve Diagnostics: Users Guide," *http://www.plantservices.com/articles/2007/017.html,* 2007. Accessed on Sep 2009.

[119] M. Germinario, "Intelligent Fluid Control Technology," *White Paper,* 2006.

[120] A. Harris and S. Schwartz, "Valves with A Brain," *Valves http://www.plantservices.com/articles/2008/121.html,* 2008. Accessed on Oct 2009.

[121] A. E. Stavale, J. A. Lorenc, and E. P. Sabini, "Development of A Smart Pumping System," *White Paper, ITT Industries, Fluid Technology Corp. -Industrial Pump Group, Seneca Falls, New York 13148,* 2001.

[122] M. Balchanos, K. Moon, N. R. Weston, and D. N. Mavris, "A Dynamic Surrogate Model Technique for Power Systems Modeling and Simulation," *SAE International,* 2008.

[123] Y. Lee, E. L. Zivi, and A. M. Cramer, "Component Heat Exchanger," *White Paper,* 2006.

[124] J. Bézivin, R. F. Paige, U. Aβmann, B. Rumpe, and D. Schmidt, "Model Engineering for Complex Systems," *Dagstuhl Seminar Proceddings,* 2008.

[125] J. R. Mayo and R. C. Armstrong, "Improving the Foundations of Complex System Modeling and Design," *Discovery at the Interface of Science and Engineering, Science Matters,* 2009.

[126] J. M. Peña, J. L. Björkegren, and J. Tegnér, "Learning Dynamic Bayesian Network Models via Cross-Validation," *Pattern Recognition Letters,* 2005.

[127] H. Lähdesmäki and I. Shmulevich, "Learning the Structure of Dynamic Bayesian Networks from Time Series and Steady State Measurements," *Machine Learning,* 2008.

[128] A. Fijany and F. Vatan, "A New Method for Sensor Placement Optimization," *41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit,* 2005.

[129] "http://www.cs.ubc.ca/~murphyk/." Accessed on Nov 2009.

[130] "http://www2.cmp.uea.ac.uk/~it/jlab/index.html." Accessed on Jun 2009.

[131] S. Müller and H. Waller, "Efficient Integration of Real-Time Hardware and Web Based Services into MATLAB," *11th European Simulation Symposium and Exhibition,* 1999.

[132] "Plug-Ins for ModelCenter®, Developer's Guide," *Phoenix Integration,* 2004.

[133] "http://jade.tilab.com/." Accessed on Jun 2009.